

AD-A085 136

INTERMETRICS INC DAYTON OH

F/8 9/3

DIGITAL AVIONICS INFORMATION SYSTEM (DAIS): MISSION SOFTWARE.(U)

FEB 80 S F STANTEN, P Y WILLIAMS

F33615-75-C-1181

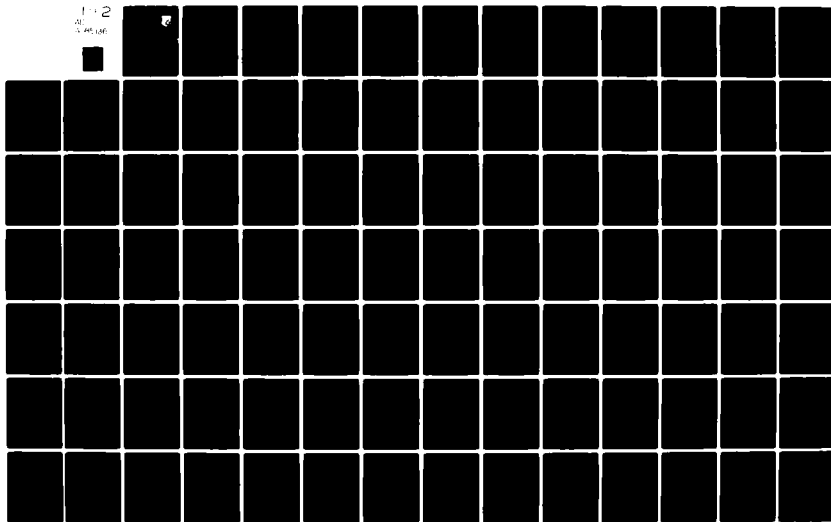
UNCLASSIFIED

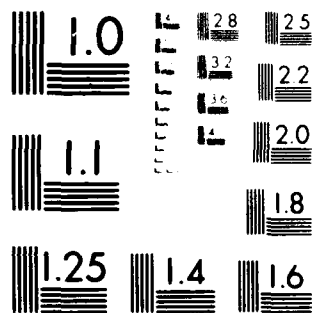
AFWAL-TR-80-1003

NL

1-2

AD-A085 136





WILSON JONES & SONS, LTD. LONDON

AFWAL-TR-80-1003

LEVEL



ADA085136

**DIGITAL AVIONICS INFORMATION SYSTEM (DAIS):**  
MISSION SOFTWARE

INTERMETRICS, INCORPORATED  
5152 SPRINGFIELD PIKE  
DAYTON, OH 45431

DTIC  
JUN 4 1980  
C

FEBRUARY 1980

TECHNICAL REPORT AFWAL-TR-80-1003  
FINAL REPORT FOR PERIOD 23 JUN 75 - 28 FEB 79

Approved for Public Release: Distribution Unlimited

AVIONICS LABORATORY  
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES  
AIR FORCE SYSTEMS COMMAND  
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

DDC FILE COPY

80 6 3 011

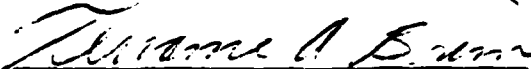
## NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.


This report has been received by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

  
Mark Thullen  
Chief Engineer  
DAIS Program Branch

  
Terrance A. Brim  
Chief, DAIS Program Branch  
Avionics Laboratory

FOR THE COMMANDER

  
Raymond E. Siferd, Col., USAF  
Chief, System Avionics Division  
Avionics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/AAAS, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) AFWAL-TR-82-1003	2. GOVT ACCESSION NO. AD-A085 236	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) DIGITAL AVIONICS INFORMATION SYSTEM (DAIS): MISSION SOFTWARE	5. TYPE OF REPORT & PERIOD COVERED FINAL REPORT 75 Jun 23 - 79 Feb 28	
7. AUTHOR(s) See reverse side.	8. CONTRACT OR GRANT NUMBER(s) F33615-75-C-1181	6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS INTERMETRICS, INCORPORATED 5162 Springfield Pike Dayton, Ohio 45431	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2092 02-03	17. 02
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE Feb 80	13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Avionics Laboratory (AFWAL/AAAS) Air Force Wright Aeronautical Laboratories Wright-Patterson AFB, Ohio 45433	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
16. DISTRIBUTION STATEMENT (of this Report)  Approved for Public Release; Distribution Unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) (9) Final rept. 23 Jun 75 - 27 Feb 79		
18. SUPPLEMENTARY NOTES (10) S.F./Stanten P.Y./Williams D./Flinn S.Z./Stein F.F./Adams		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Application Software	Displays	Operational Flight Programs
Avionics Software	Executive	Processor
Avionics System	Higher Order Software	Real-Time
Computers	JOVIAL	Simulation
DAIS	Multiplex	Software Development
	Operating Systems	Software Standards & Regulations
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The significance of this research and development to the Air Force is the demonstration of the applicability of modern software techniques and methodology to the generation of sophisticated, flexible avionics software with excellent life cycle costs and easy maintainability. The DAIS Mission Software was coded in J73/I and based its structure upon the relocatability concepts known as Higher Order Software (HOS).		

411731

## 20. ABSTRACT

The resultant Digital Avionics Information System (DAIS) Mission Software has been demonstrated for two different missions based respectively upon an A-7 and an A-10. An interface between the Application Software and the Executive System has been developed from this interface requirement. In addition, the DAIS Mission Software effort has extended knowledge: in the software processing of federated computer systems; the developing of Application Software to be later partitioned; the automatic generation of interprocessor I/O communicating; and demonstrated the feasibility and simulation techniques required for the cost effective and reliable development of an Operational Flight Program (OFF) on a host computer system, thus relieving the problem of limited access and limited capability of a unique target computer facility.

Author(s):

W. H. Vandever, Jr.  
S. F. Stanten  
P. Y. Williams  
D. A. Flanders  
S. Z. Stein  
S. E. Adams  
N. Eastridge

## FOREWORD

This report is submitted in partial fulfillment of the final report, CDRL item A003, for the DAIS Mission Software, F33615-75-C-1181. Intermetrics would like to express its pleasure in participating in the DAIS effort. This cooperative and evolving effort combined many industrial and governmental participants. It has illustrated the ability of the Air Force to respond in a timely fashion to the dynamic environment of modern technology. AFAL is to be congratulated in their successful development and management of this program.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution _____	
Available for _____	
Dist. _____	
A	

## TABLE OF CONTENTS

<u>SECTION</u>	<u>PAGE</u>
I INTRODUCTION AND SUMMARY . . . . .	1
II BACKGROUND . . . . .	4
2.1 Avionics Software. . . . .	5
2.2 DAIS Baseline. . . . .	8
2.2.1 DAIS Hardware. . . . .	8
2.2.2 The Basis for the DAIS Application . . . . .	9
2.2.3 Software Test Stand (STS) and Integrated Test Bed (ITB) . . . . .	9
2.2.4 DAIS Software Elements . . . . .	9
III DESIGN OBJECTIVES AND METHODOLOGY. . . . .	10
3.1 Software Structure . . . . .	10
3.2 Avionic Executive Implementation Considerations. . . . .	16
3.2.1 Well Defined Set of Mission Software . . . . .	18
3.2.2 Real-Time Data Conflicts . . . . .	18
3.2.3 Real-Time Criticality. . . . .	19
3.3 Use of JOVIAL J73/I. . . . .	19
3.3.1 JOVIAL Central Statement Restrictions. . . . .	20
3.3.2 Built-in Functions . . . . .	21
3.3.3 Real-Time Interface. . . . .	22
3.4 Avionic Environment. . . . .	24
3.5 Managerial Control: DAIS Software Standards . . . . .	25
3.5.1 Programming Standards. . . . .	29
3.5.2 Documentation Standards. . . . .	30
IV TECHNICAL DESIGN . . . . .	31
4.1 Executive Software Overview. . . . .	31
4.2 Applications Software Overview . . . . .	37
4.3 Applications/Executive Interface . . . . .	42
4.3.1 Real-Time Declaration. . . . .	44
4.3.1.1 Task Declarations. . . . .	44
4.3.1.2 Event Declarations . . . . .	45
4.3.1.3 Compool Block Declarations . . . . .	45
4.3.2 Real-Time Statements . . . . .	47
4.3.2.1 Schedule Statements. . . . .	48
4.3.2.2 Cancel Statements. . . . .	49
4.3.2.3 Terminate Statements . . . . .	49



## TABLE OF CONTENTS (Cont'd)

<u>SECTION</u>	<u>PAGE</u>
4.3.2.4 Wait Statements . . . . .	50
4.3.2.5 Signal Statements . . . . .	53
4.3.2.6 Read Statements . . . . .	53
4.3.2.7 Write Statements. . . . .	53
4.3.2.8 Trigger Statements. . . . .	54
4.3.2.9 Access Statements . . . . .	55
4.3.2.10 Broadcast Statements. . . . .	55
4.3.2.11 Forced Read Statements. . . . .	56
4.3.3 Real-Time Built-in Functions. . . . .	56
4.3.3.1 Event Read Functions. . . . .	56
4.3.3.2 Task Event Read Function. . . . .	57
4.3.3.3 Time Read Function. . . . .	57
4.3.3.4 Minor Cycle Read Function . . . . .	57
4.3.4 Real-Time Directives. . . . .	58
4.3.4.1 Privileged Mode Directive . . . . .	58
4.3.4.2 Local Override Directive. . . . .	59
4.4 Executive System. . . . .	59
4.4.1 Introduction. . . . .	59
4.4.1.1 Synchronous Actions . . . . .	62
4.4.1.2 Asynchronous Actions. . . . .	63
4.4.2 DAIS Executive Functional Description . . . .	63
4.4.2.1 Local Executive . . . . .	64
4.4.2.2 Master Executive. . . . .	64
4.4.3 Interfaces With Real-Time Software. . . . .	67
4.4.3.1 General Overview. . . . .	67
4.4.3.2 Tasks . . . . .	67
4.4.3.2.1 Task States . . . . .	67
4.4.3.2.1.1 INVOKED/UNINVOKED . . . . .	68
4.4.3.2.1.2 ACTIVE/INACTIVE . . . . .	68
4.4.3.2.1.3 WAITING/DISPATCHABLE . . . . .	71
4.4.3.2.1.4 READY/SUSPENDED/ EXECUTING . . . . .	71
4.4.3.2.2 Task Hierarchy. . . . .	71
4.4.3.2.3 Priority. . . . .	72

# TABLE OF CONTENTS (Con't)

<u>SECTION</u>	<u>PAGE</u>
4.4.3.2.3.1 Normal Mode Tasks .	72
4.4.3.2.3.2 Privileged Mode Tasks . . . . .	72
4.4.3.3 Comsubs . . . . .	73
4.4.3.4 Compool Blocks. . . . .	73
4.4.3.4.1 Local Copies. . . . .	74
4.4.3.4.2 Categories of Compool Blocks.	75
4.4.3.4.2.1 Synchronous Compool Blocks. . . . .	75
4.4.3.4.2.2 Asynchronous Compool Blocks. . . . .	77
4.4.3.4.2.3 Critically Timed Compool Blocks. . .	77
4.4.3.4.2.4 Minor Cycle Tag Words . . . . .	77
4.4.3.5 Events. . . . .	77
4.4.3.5.1 Event Types and Usage . . .	78
4.4.3.5.1.1 EVENT . . . . .	79
4.4.3.5.1.2 SCHEDULE. . . . .	79
4.4.3.5.1.3 WAIT. . . . .	81
4.4.3.5.1.4 SIGNAL. . . . .	81
4.4.3.5.1.5 EREAD . . . . .	81
4.4.3.5.1.6 INVOKED . . . . .	82
4.4.3.5.2 Event Condition Set . . . .	82
4.4.3.6 Time. . . . .	83
4.4.3.7 Real-Time Interfaces. . . . .	84
4.4.3.7.1 Privileged Mode Directive .	84
4.4.3.7.2 Local Copy Override . . . .	85
4.4.3.8 Master Executive Interfaces . .	85
4.4.3.8.1 Startup/Loader. . . . .	85
4.4.3.8.2 Master Initialization . . .	86
4.4.3.8.3 Bus Control Interfaces. . .	91
4.4.3.8.4 Monitor-Backup Interfaces .	91
4.4.3.8.5 System Configuration Management Interface . . . . .	94
4.4.3.8.6 Reconfiguration Interface . .	94

## TABLE OF CONTENTS (Cont'd)

<u>SECTION</u>	<u>PAGE</u>
4.4.3.8.7 Mass Memory Control Interface . . . . .	96
4.4.4 Hardware Interfaces . . . . .	97
4.4.5 Interface to PALEFAC. . . . .	97
4.5 Applications Software . . . . .	98
4.5.1 Equipment Interface . . . . .	98
4.5.2 Software Interface. . . . .	98
4.5.3 Application Software Architecture . . . . .	101
4.5.3.1 System Control Modules. . . . .	101
4.5.3.2 Specialist Functions (SPECS). . . . .	102
4.5.3.3 Equipment Processes (EQUIPs). . . . .	103
4.5.3.4 Display Processes (DISPs) . . . . .	104
4.5.4 Software Interactions . . . . .	104
4.5.4.1 IMFK/MFK - Pilot Interface. . . . .	104
4.5.4.2 Navigation Interfaces . . . . .	108
4.5.4.3 Normal Attack Interactions. . . . .	109
4.5.4.4 Interaction of Input EQUIP Functions	110
V DEVELOPING A MISSION. . . . .	111
5.1 Mission Development . . . . .	113
5.2 Simulation and Testing. . . . .	116
5.2.1 Simulation Capabilities . . . . .	116
5.2.2 Simulation Method . . . . .	119
5.2.3 Simulation Tools. . . . .	123
5.2.3.1 General Level Ø Testing . . . . .	125
5.2.3.2 Level Ø Testing of Display Software	126
5.2.3.3 Level 1 and Level 2 Testing . . . . .	126
5.2.3.3.1 TOMBS . . . . .	126
5.2.3.3.2 TOMBS and AVSIM . . . . .	128
VI MISSION SOFTWARE ILLUSTRATIONS. . . . .	129
6.1 Mission A to Mission α. . . . .	129
6.2 Non-DAIS Device Protocol. . . . .	130
6.3 Throughput Optimization . . . . .	132
6.4 Proof of Concept. . . . .	133
6.5 Use of J73/I. . . . .	134
6.6 Embedded Performance Monitor. . . . .	135
VII CONCLUSION AND RECOMMENDATIONS. . . . .	138
BIBLIOGRAPHY. . . . .	142

## LIST OF FIGURES

<u>NUMBER</u>	<u>TITLE</u>	<u>PAGE</u>
1	Hierarchical Control Structure . . . . .	13
2	Operational Flight Program Software. . . . .	32
3	Executive (System) Software. . . . .	33
4	Executive Structure Breakdown. . . . .	36
5	Applications Software. . . . .	38
6	DAIS System Layers . . . . .	41
7	DAIS System Architecture . . . . .	61
8	Task States and Control. . . . .	68
9	Task State Transition Diagram . . . . .	70
10	Startup/Loader Interface . . . . .	87
11	Master Initialization Interface. . . . .	88
12	Bus Control. . . . .	89
13	Bus Control in Remote Processors . . . . .	92
14	Monitor/Recovery Interface . . . . .	93
15	System Configuration Management Interface. . . . .	95
16	Mass Memory Control Interfaces . . . . .	96
17	IMFK/MFK Interactions. . . . .	107
18	Development of Mission Software. . . . .	113
19	Building DAIS Mission Software . . . . .	114
20	System Checkout Configuration . . . . .	122

# LIST OF TABLES

<u>NUMBER</u>	<u>TITLE</u>	<u>PAGE</u>
1	Program Contributors . . . . .	3
2	Basic Structure Design Standards . . . . .	17
3	JOVIAL J73/I Control Statements. . . . .	21
4	DAIS Functions and Definitions . . . . .	22
5	DAIS Executive Real-Time Constructs. . . . .	23
6	DAIS Commonly Used Subroutines (Comsubs) .	26,27
7	Categories of Compool Blocks . . . . .	76
8	Events: Types and Their Usage . . . . .	79
9	Result of Privileged Mode Directive. . . . .	85
10	Mission & Configuration. . . . .	99

## SECTION I. INTRODUCTION AND SUMMARY

The Air Force Digital Avionic Information System (DAIS) Program had as its object the design, development and specification of a standardized, modular, reliable digital avionics system with easy maintainability and low life cycle costs. The modularity concepts included both hardware modularity, i.e., variable numbers of processors and subsystems, and software modularity, i.e., mission dependent packages composed of software building blocks. Central to the achievement of these architectural characteristics was the development of an applications independent executive function which could efficiently integrate the diverse hardware requirements and applications software into a cohesive, efficient unit. The executive software and the modular applications software are collectively known as the DAIS Mission Software and are the subject of this report.

As in any program of significant magnitude, the DAIS program had the benefit of many elements and individuals within the aerospace community. A few of the more significant participants in the DAIS program and their contributions to the efforts are given in Table 1. It should be noted that the DAIS program has evolved during its active history from 1973 to the present. It will undoubtedly continue its evolutionary growth, lending significant contributions to future avionics system concepts and designs.

The remaining sections of this report discuss some of the more significant details of the DAIS Mission Software. A general guide to the report is as follows:

Section 2.0 BACKGROUND - A general overview of the antecedent software problems associated with avionics systems and a brief description of the DAIS system hardware and software environment.

Section 3.0 DESIGN OBJECTIVES AND METHDOLOGY - A discussion of the attendant framework of avionics system software development problems and the design solutions that evolved.

Section 4.0 TECHNICAL DESIGN - A description of the details of the technical design of the executive and application software and their interaction.

Section 5.0 DEVELOPING A MISSION - A brief overview of the production environment used to develop DAIS Mission Software.

Section 6.0 MISSION SOFTWARE ILLUSTRATIONS - An account of the evolutionary growth of the DAIS Mission Software and examples of the attributes of the software design.

Section 7.0 CONCLUSIONS AND RECOMMENDATIONS

The DAIS Mission Software was programmed using the Higher Order Language (HOL) JOVIAL J73/I in conjunction with modern structured software techniques and relocatability concepts derived from Higher Order Software (HOS) principles. The results of this effort have been the successful development and demonstration of the application software for two close air support missions: one based on an A-7 aircraft and the other on an A-10 aircraft. As part of this effort a real-time interface between the applications software and the executive software was specified. The resultant executive software has been successfully used on the Single Seat Attack program and forms the baseline for an Air Force standardized Executive.

The success of the Mission Software effort shows the applicability of both modern software techniques and integrated design to digital avionics system applications.

Table 1. Program Contributors

<u>Organization</u>	<u>Contribution</u>
AFAL	Program Management
Charles Stark Draper Laboratory	Design Consultants
Computer Sciences Corporation	JOVIAL J73/I Compilers
General Dynamics	Preliminary Design Study
Hughes Aircraft Company	Controls and Displays
International Business Machines	Multiplex Equipment
Intermetrics Incorporated	Mission Software
Texas Instruments	Preliminary Design Study, and Initial Hardware and Software Specifications
The Boeing Company	Initial Hardware and Software Specifications
TRW	System Integration and Test Coordination
Westinghouse	DAIS Processors: AN/AYK-15



## SECTION II. BACKGROUND

The Air Force has been increasingly concerned with the rising costs of developing and maintaining weapon systems. A significant contributor to this rise in cost is the continuing development of unique and singular systems and software designs for each specific application. In countering this trend, the Air Force has applied both technical and managerial techniques to stop the need for this proliferation.

The problem stems from the fact that the role of digital computers in new weapon systems continues to expand. Even today weapon systems of similar processing requirements possess widely differing computer system designs, differing software tools and philosophy, and correspondingly different support tools and requirements.

One of the more important programs directed toward a solution to this problem is the Digital Avionics Information System (DAIS). The DAIS program has been an ongoing effort at the Air Force Avionics Laboratory (AFAL) since 1973 to develop tools and techniques for producing reliable, effective systems with more desirable attributes than previously available. It has developed from a conceptual phase through a design phase to an actual implementation of a system without the high life cycle costs of previous designs. The current implementation is scheduled to be refined and then to be transitioned to the production environment of the Aeronautical Systems Division (ASD).

The Air Force DAIS Program has been concerned with long term life cycle cost issues for avionics, and has addressed them within the context of currently available technology. DAIS has systematically integrated and applied modern technology and techniques to the avionics problem. Significant among the technologies incorporated in DAIS are the use of

the MIL-STD-1553A multiplexed data bus, a generalized federated computer system architecture for reliability and for computational growth, the use of general purpose programmable pilot displays, and the specification of modern software engineering practices.

The DAIS program has successfully incorporated such technologies and has demonstrated their applicability to military avionics systems. The DAIS software has been implemented in the HOL Jovial J73/I and structured according to design standards devised for reliability, maintainability and low life cycle costs. An important part of this development has been the implementation of a DAIS Executive System which minimizes the need for applications programmers to be concerned with the details of multi-processing and I/O hardware operation.

## 2.1 Avionics Software

In the long term, the software cost of an avionics system is dominated by the modification of, additions to, and maintenance of the mission oriented application software, i.e., the Operational Flight Program (OFP). Originally, digital avionics processors were programmed in assembly language. The resulting code was usually hand crafted to obtain the desired performance characteristics. Modification of such OFPs was difficult and costly and the addition of any new feature often required both a major redesign and a compromise to current features. Consequently, modern avionics systems such as the Air Force B-1 and F-16 programs specified the use of the Higher Order Language (HOL) J3B for their avionic systems. Similarly, NASA selected the higher order language HAL/S for programming the Space Shuttle. The use of a HOL narrows the range of possible programming errors, and greatly improves the comprehensibility and maintainability of the resultant OFP. Additionally the use of a HOL frees the OFP programmer from the specific characteristics of a given

target machine, making feasible initial code and development and module testing on a general purpose computer. This technique has the advantages of easier facility accessibility, earlier development, and the availability of more extensive diagnostic tools for software development than is usually (if at all) available with typical target military computers.

One of the major achievements of the DAIS Mission Software development effort has been the design, specification and development of a standard avionics executive. The advantages of standardization include the reduction of development errors due to wider applicability (and thus legacy), a reduction in training and documentation costs and portability. The DAIS executive has greater than usual flexibility for avionics applications because it accomodates certain real-time functional characteristics which are a common requirement, regardless of the particular mission or aircraft. These real-time characteristics typically are required to compute navigation, guidance, and weapon delivery information; to read and control sensors and actuators; to communicate with the pilot via cockpit controls and displays; and to control both the periodic and asynchronous execution of these functions. These functional characteristics include real-time process interaction: initiation of processes, repetitively, upon the occurrence of some event or at some specific time; the conditional execution of processes; the ability to monitor the real-time process itself; and communication with the actual I/O devices. It is this set of real-time capabilities that the DAIS executive has successfully incorporated. A further refinement has been the implementation of these real-time executive services via the JOVIAL DEFINE capability in a manner which appears to the programmer as a language augmentation.

The DAIS executive and its interface to the application software has been designed to afford both straightforward design and reliable modification of the application software. This has in part been accomplished by raising the level of the user's interface to the applications software. It makes the application software invariant with respect to the computer, the computer network, the I/O implementation, and the implementation of the executive itself. This has been done by creating a "logical" interface between the executive and the application software: the process and I/O control has been represented by a well defined standard set of language primitives. It has therefore become possible to develop and maintain OFF software in a logical, application oriented fashion, without the heretofore degenerating effect of system dependencies. It has become feasible to achieve a common standard across the total active fleet and to obtain benefits with respect to personnel qualifications, training, documentation, software support cost, and the sharing of expertise.

Careful attention has been paid to the structuring and design of the application software in DAIS. Much of the current interest in structured programming and other software engineering techniques does not concern itself with real time software, and most improved programming techniques are not applicable to real time problems. Real time software has severe performance and time constraints. It has introduced a new dimension called "time". The DAIS Mission Software baseline design was an outgrowth of research into Higher Order Software (HOS). HOS was concerned with the application of modern software design and structuring techniques to real-time software. The adaptation of abstract HOS concepts to actual software practice was initially undertaken for DAIS by the Charles S. Draper Laboratory. It was developed and applied by Intermetrics to the design of the Mission Software. This mapping of HOS concepts into software reality resulted in the preliminary design for the DAIS Executive.

## 2.2 DAIS Baseline

While DAIS has addressed avionics software development in general, it necessarily consists of a specific set of elements in its current implementation. The current system employs a particular set of hardware, oriented to a specific aircraft and mission, having a specific set of avionics equipment, and is provided with specific software support elements.

### 2.2.1 DAIS Hardware

The elements of the DAIS hardware configuration which are relevant to the Mission Software are the processors, the data bus, the controls and displays systems, and the various interfacing equipment.

DAIS contains one or more AN/AYK-15 processors organized as a federated system. One, two, and three processor systems have been tested and demonstrated. Each processor within the system possesses its own dedicated memory and executes a set of prepartitioned software modules. Software processes in different computers communicate over a common data bus interconnecting all of the processors and all other system elements.

DAIS incorporates a dually redundant data bus based on MIL-STD-1553A. DAIS has developed a detailed bus protocol for its baseline implementation, although there is provision for other (non-DAIS) protocols.

DAIS also incorporates a set of general and multi-purpose programmable displays, which have undergone evolutionary changes during the program. The current display system consists of five CRT displays, switchable refresh memories, programmable symbol generators, scan converters, and some dedicated instrument displays. The particular set of Control and Display equipment is not critical to either the functioning or design of the DAIS Mission Software.

#### 2.2.2 The Basis for the DAIS Application

The present DAIS application called, Mission Alpha ( $\alpha$ ), has been modeled after the A-10 aircraft. While the current A-10 fleet aircraft has not been assigned a digital avionics system, DAIS has specified sets of equipments consistent with the Close Air Support Missions of the A-10. The original DAIS Mission Software Application, called Mission A, was based upon the A-7D but was later modified to its current A-10 configuration.

#### 2.2.3 Software Test Stand (STS) and Integrated Test Bed (ITB)

The DAIS STS and ITB are combined hardware/software facilities, consisting of both real and simulated equipments. The ITB is the larger system and contains:

- actual DAIS AN/AYK-15 processors
- actual data bus system including Bus Control Interface Units (BCIU) and Remote Terminals
- actual control and display equipment
- a cockpit simulator driven from the DAIS Hot Bench which consists of a DECsystem-10 and a number of PDP-11s used to simulate the environment, the aircraft, and various equipments
- other elements of hardware and software required to simulate, buffer, record and control the ITB system.

#### 2.2.4 DAIS Software Elements

The primary software support tool developed by DAIS has been the JOVIAL J73/I compiler hosted on the DECsystem-10 and targeted for the DECsystem-10 and the AN/AYK-15. The AN/AYK-15 is supported by the DAIS ALAP assembler, DAIS LINKS linker, and DAIS ASYTRAN loader.

### SECTION III. DESIGN OBJECTIVES AND METHODOLOGY

The design objectives and methodology used to develop the DAIS Mission Software were based on several different concepts and techniques. Besides the use of the modern concepts and techniques of structural programming and top-down design, DAIS based its design principles upon additional concepts espoused by the Higher Order Software (HOS) study. Another aspect of the methods used involves the applicability of HOS concepts to the JOVIAL J73/I language used by DAIS. This mapping of abstract concepts into practice entailed both specific development and modifications in detail of the basic HOS principles. A third aspect of the DAIS design objectives and methods was with respect to an efficient and optimized software package oriented specifically to the avionics environment in contra-distinction to a ground based computer network. Finally, an extremely important aspect of the DAIS design was the development of a set of software standards to both communicate the design and control the specific implementation. The human comprehension aspect of a large software project can be never be neglected; proper design and implementation cannot be accomplished if all the individuals working on the project are not properly disciplined in a unified design approach.

#### 3.1 Software Structure

One of the objectives of DAIS was to use the latest software technology applied in a systematic and integrated fashion in order to demonstrate its applicability to the development of avionics systems. The DAIS Mission Software baseline was established upon the research work performed in Higher Order Software (HOS). This research was

concerned with the application of modern software design and structuring techniques to real-time software. Software engineering techniques do not usually address real-time software problems and are not always applicable to real-time problems. Real-time software has severe performance and time constraints and introduces the new dimension called time.

The application of abstract HOS concepts to actual software practice was initially undertaken for DAIS by the Charles S. Draper Laboratory. This mapping of HOS concepts into practice formed the preliminary design for the DAIS Executive. Conceptually, HOS principles try to remove real-time problems as found in avionics software by enforcing structured software standards on this set of problems.

The DAIS Mission Software design process can be viewed as a three step process. The first step was the functional breakdown of the mission requirements in a systematic top-down fashion. This initial step provides a functional decomposition of the mission requirements into smaller, logically complete and more detailed functional specifications.

The second step was to structurally reorganize these functional specifications in a manner which would expose real-time software problems and thus allow their solution. The methodology was based upon the theoretical work collectively known as Higher Order Software. These rules require a correlation of real-time criticality with priority structure and the creation of sequential modules in order to structure response requirements. The benefits gained through this discipline are:

- The removal of certain real-time problems by means of structural definition (e.g., data protection).



- The comprehension of potential dynamic interactions in a static fashion (e.g., through planar tree graphs of the priority structure).
- The necessary hierarchy of the various functions and their interactions (e.g., it exposes absolute priority, shared data, required re-entrancy).
- Potentially, the deduction by static means of various worst case situations (e.g., maximum execution time).

The third step was to provide the Specification Design, i.e., the detailed design. It was extremely important to differentiate between the Requirements and Specification Design steps. Requirements Design was concerned with functional operation, software structure, definition of control and calculation modules, and identification of data interfaces and error conditions. Specification design addresses the implementation, taking into account the programming language, e.g., JOVIAL J73/I, the details of the executive, the avionics system configuration and the demands of efficiency. Here efficiency must be viewed within the context of the total software development: execution time, memory space, growth potential, maintainability and documentation readability.

To the greatest degree possible, the DAIS Mission Software was organized in a hierarchical control tree structure. All applications software consisted of either Controller or Calculator functions. Every Calculator must be controlled by a unique Controller. (Common service routines are excepted in that they may be invoked from many program modules.) As a requirements design principle, control over modules conform to this particular hierarchy. A module, therefore, can only invoke modules on its next lower level, not on the same level, and cannot invoke itself. Each mode (marked N in Figure -1) represents a Controller with a sequence, or choice, of functions

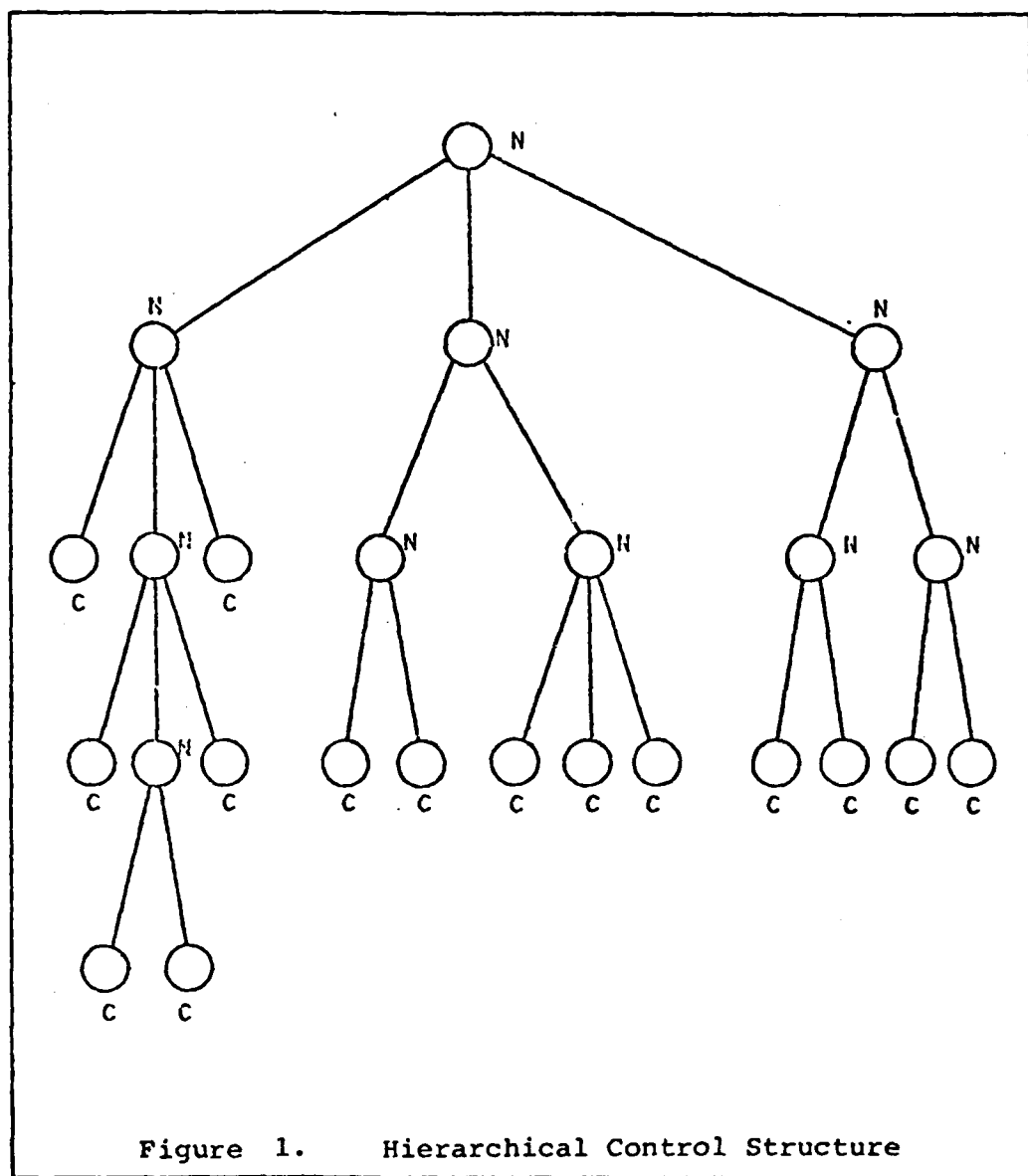


Figure 1. Hierarchical Control Structure

it controls. Each Calculator module (marked C) represents a data manipulation function providing for no further control.

The control hierarchy implies a descending process priority in the sense that the local executive will preempt lower priority processes in favor of pending processes of higher priority.

In order to reduce the number of possible logic paths and thereby facilitate the checkout and verifications of DAIS Mission Software, all modules (Controller and Calculator) were designed to be single purpose. That is, extraneous functions were not performed if the by-products were not intended as an output. In addition, multi-mode modules were avoided, i.e., those which change their function significantly based on the internal examination of flags. This was tempered with due consideration for speed and size efficiencies.

In order to increase the degree of determinism of the hybrid (synchronous/asynchronous) real-time system which included data protection and restartability, a strict set of requirements design rules for order control was required as follows:

- A Controller establishes the priority, timing and sequence, and conditions for the execution of the processes on the next lower hierarchical level and only that level. This ordering is established by requests to the executive function.
- The priority of a Controller is higher than the priority of any process on its most immediate lower level. Higher priority processes may be activated by a signal from a lower priority process.
- If two processes have the same Controller and the first has a lower priority than the second, then all processes in the control structure of the first are of lower priority than the processes in the

control structure of the second. A process that may be preempted by another process, may be preempted by any of the other process' sub-processes.

- Processes are activated when a set of event conditions are satisfied. The set of event conditions is selected by the controller of the process, but the events tested in the condition may be signalled ON or OFF by any process. This relieves the Controller of the need to monitor for the condition and activate the process at the proper time.
- All data interfaces between processes are communicated through Compools by the executive system. This is accomplished by effective "READ/WRITE" requests. Executive system intervention is required to insure data integrity across all DAIS processors. From a program point of view I/O data will be similarly treated and accessed within the Compools by the READ/WRITE requests.

While these principles appear to be abstract, in the DAIS context they were mapped into a finite set of basic building blocks. These are four building blocks for the DAIS Mission Software from the application software point of view. They are:

- Compool Blocks
- Tasks
- Comsubs
- Events

Compool Blocks represent global data and are centrally defined and controlled for a given project. Tasks are the real-time processes within the system. Tasks pass information solely via Compool Blocks and must follow the DAIS HOS structuring rules. Comsubs are commonly used subroutines.

These subroutines perform calculations only and have no real-time control or interaction. Comsubs receive and send information solely via parameter passage and have no access to Compool Blocks. Due to the multiprocess nature of a real-time system, there is the implied requirement for re-entrancy with Comsubs. Events are binary valued control information that enable tasks and provide the environment for tasks to interact. Within the DAIS system, Events may be considered to be either "latched", i.e., ON or OFF, or "unlatched", i.e., a pulse, either plus or minus.

Using these building blocks, the HOS derived DAIS baseline primarily effects the structuring and interrelationships of Tasks. The major structuring requirements are summarized in Table 2.

The DAIS Executive has taken some variance with respect to this set of structuring rules. While the software was developed within these objectives, some other features have been added to allow an escape from this rigid structure. This need arises either because of real-time criticality or efficiency considerations. These modifications allow for:

- The specification of an absolute priority
- The ability to wait on time or an event
- The ability to update a Compool Block within a single Task.

### 3.2 Avionics Executive Implementation Considerations

Avionics software differs from most ground based software in ways that can impact design and implementation decisions. This includes the possibility for optimization because of a well-defined set of mission software, the concern with real-time data conflicts, and the real-time criticality of the system.

## TABLE 2

### BASIC STRUCTURED DESIGN STANDARDS

#### DAIS BASELINE: HOS DERIVED

- TASKS ORDERED BY PRIORITY.
- PLANAR TREES OF PRIORITY THAT CAN BE LINEARIZED.
  - ROOT OF TREE HAS HIGHEST PRIORITY, AND THE OFFSPRING OF A CONTROLLER ARE STRICTLY ORDERED: IF  $\text{PRIO}(A) > \text{PRIO}(B)$  THEN FOR ALL DESCENDENTS  $A'$  AND  $B'$  OF  $A$  AND  $B$ ,  $\text{PRIO}(A') > \text{PRIO}(B')$ .
- CONTROL OF A MODULE IS SOLELY DEPENDENT BY SPECIFICATION ON ITS CONTROLLER.
  - EVENTS AND/OR TIME MAY CAUSE A MODULE TO EXECUTE
- DATA STRUCTURE.
  - DATA INTO AND OUT OF A MODULE IS KNOWN BY ITS CONTROLLER.
- MODULE CONSTRUCTION.
  - ONE FUNCTION.
  - NO KNOWLEDGE OF CONTROLLER.

### 3.2.1 Well Defined Set of Mission Software

In the case of the avionics software, the total set of software is known pre-run time, thus allowing for various static instead of dynamic implementations. For example, data structure organizations can include indices versus pointers since indices are in general more efficient both in space and execution time. The number of run-time parameters for executive routines can also be minimized if the information is built directly into the executive tables. In addition, it becomes possible to analyze possible and allowable task and data interactions.

One of the major concerns DAIS had was with respect to data efficiency. DAIS takes full advantage of static data optimization possibilities by providing a pre-run time executive table generator and analyzer. This tool Partitioning Analysis, and LinkEdit FACility (PALEFAC), is an integral part of the DAIS Executive systems and allows for optimized executive data information.

### 3.2.2 Real Time Data Conflicts

Data conflicts can occur in real-time systems due to several different circumstances. Multiple updates of information caused by such events as Direct Memory Access of I/O can lead to non-homogeneous data. Accessing arrays of data presents a problem with respect to assuring data concurrence since an array can be partially modified while it is being accessed.

These data conflicts can be solved in various fashions including explicit task and I/O design and layout; by explicit use of Update Blocks, e.g., HAL/S; by implicit I/O - data interlocks; by the implicit use of double buffering in tasks or some combination of the above. Ideally, an implicit solution would be desirable in order to

eliminate the problem so that from the application programmer's point of view, it could not occur. This is indeed the desire and design philosophy championed by HOS. Within the DAIS system a double buffering method was implemented but with more efficient alternate methods allowable in certain cases.

### 3.2.3 Real Time Criticality

Avionics executives must be concerned with both the time accuracy of the I/O interface signals and CPU saturation. In a weapons system, for example, the accuracy of an unguided weapon is dependent upon an accurate time of release. Additionally, it is not an acceptable practice to allow an avionics executive to stop; the system must continue. One major area of real-time design criticality for the DAIS Mission Software was the integrated development of restart and reconfiguration capabilities as part of the overall software package. In addition, CPU throughput considerations cause revisions to the HOS principles to bring them into conformance with the practicalities of actual system implementations.

### 3.3 Use of JOVIAL J73/I

The programming language selected for the DAIS Mission Software was JOVIAL J73/I. The use of J73/I combined with the design methodology espoused by DAIS required both control of and modification to J73/I constructs and the addition of several other features not available with J73/I. The use and appearance of JOVIAL for the DAIS Mission Software was modified in several respects.

- The use of some JOVIAL language features were restricted, prohibited, or modified with respect to certain of its constructs.
- Additional "built-in functions" were provided to the DAIS Mission Software programmer.



- A Real-Time interface was provided that augments the language available to the Mission Software programmer.

In addition, various commonly used procedures were made available for programming the DAIS Mission Software.

### 3.3.1 JOVIAL Control Statement Restrictions

An example of types of restrictions imposed by DAIS can be found with the J73/I control statements. Control statements are executable statements that alter the normal flow of a program or subprogram. Normally, a program begins with the execution of the first executable statement in the program. When the execution of that statement is completed, the next sequential executable statement is executed. This process continues until the program ends. A subroutine, when referenced, starts with its first executable statement, then executes the next sequential executable statement, and so on, until it returns control to the program statement that referenced it. Control statements alter this sequential flow.

In structured programming, emphasis is placed upon the visible and orderly transfer and return of control. JOVIAL contains some control statements that work against the intent of this structured-programming philosophy. Therefore, the use of some control statements is either prohibited, restricted, or actively discouraged by good structured-programming practice. Table 3 lists the JOVIAL J73/I control statements and their status with respect to DAIS Mission Software usage.

Table 3. JOVIAL J73/I Control Statements

Usage \ Jovial Control	GOTO, STOP, RETURN, IF, SWITCH, WHILE, FOR, Procedure Call
Forbidden Usage	STOP
Discouraged Usage	GOTO
Restricted or Modified Usage	SWITCH, FOR, Procedure Call
Standard Usage	RETURN, IF, WHILE

### 3.3.2 Built-In Functions

The J73/I language does not define a standard algorithm or function package as part of the language specifications. It is obvious that such functions as SIN (sine) and COS (cosine) are required for avionics programming. DAIS developed and specified a set of such functions and these are delineated in Table 4. There was one other major design requirement with respect to these functions. The mission software environment is that of a multiprocess and multiprocessors. It is therefore necessary to make these arithmetic functions appear to be reentrant. Reentrancy is not directly supported in J73/I, but rather the language has the concept of "based" procedures. This in turn requires the appearance of a break point in a routine invocation. This provision for the built-in functions was done in such a fashion as to provide the user with function calls of the forms:

SIN @PTR (THETA) or  
SIN (PTR, THETA).

The underlying design principle was to present to the application programmer the interface to which he is familiar and which he desires.

Table 4. DAIS Functions and Definitions

Function	Definition
ACOS (argument)	Arc cosine
ASIN (argument)	Arc sine
ATAN (argument)	Arc tangent
ATAN2 (arg1,arg2)	Arc tangent of arg1/arg2
COS (argument)	Cosine
COSH (argument)	Hyperbolic cosine
EXP (argument)	Exponential function
LN (argument)	Natural logarithm
LOG (argument)	Logarithm to the base 10
MAX (arg1,arg2)	Maximum of arg1 and arg2
SIN (argument)	Sine
SINH (argument)	Hyperbolic sine
SQRT (argument)	Square root
TAN (argument)	Tangent
TANH (argument)	Hyperbolic tangent

### 3.3.3 Real-Time Interface

The DAIS methodology contains both solutions to the real-time data update problems and also provides for a clean methodology of software restarts. The structuring is predicated on the fact that references to global data be in actuality references to a local copy of the global data. In certain circumstances, where Tasks are of short duration and of highest priority, reference to the actual "global copy" of the data is allowed. Table 5 describes the DAIS real-time executive constructs which provide the task real-time interfaces.

Table 5 DAIS EXECUTIVE REAL TIME CONSTRUCTS	
<p><b>PROCESS CONTROL STATEMENTS</b></p> <ul style="list-style-type: none"> <li>• SCHEDULE (<u>&lt;task name&gt;</u>, PRIORITY=<u>&lt;prio&gt;</u> {<u>&lt;unlatched&gt;</u>} {<u>&lt;latched&gt;</u>} {<u>&lt;time field&gt;</u>});  <u>&lt;prio&gt;</u> ::= <u>&lt;integer&gt;</u>   -<u>&lt;integer&gt;</u>  <u>&lt;unlatched&gt;</u> ::= ,UPON<u>&lt;event expression&gt;</u>  <u>&lt;latched&gt;</u> ::= ,IF<u>&lt;event expression&gt;</u>  <u>&lt;time field&gt;</u> ::= ,PHASE=<u>integer</u>, PERIOD=<u>integer</u>  <u>&lt;event expression&gt;</u> ::= <u>&lt;factor&gt;</u>   <u>&lt;factor&gt;</u> AND <u>&lt;event expression&gt;</u>  <u>&lt;factor&gt;</u> ::= [<u>&lt;factor2&gt;</u>]   <u>&lt;factor2&gt;</u>  <u>&lt;factor2&gt;</u> ::= <u>&lt;or set&gt;</u> = ON   <u>&lt;or set&gt;</u> = OFF   <u>&lt;or set&gt;</u>  <u>&lt;or set&gt;</u> ::= <u>&lt;event&gt;</u>   <u>&lt;event&gt;</u> OR <u>&lt;or set&gt;</u></li> <li>• CANCEL (<u>&lt;task name&gt;</u>);</li> <li>• TERMINATE (<u>&lt;task name&gt;</u>);</li> <li>• WAIT UNTIL (<u>&lt;time&gt;</u>);</li> <li>• WAIT FOR (<u>&lt;time&gt;</u>);</li> <li>• WAIT (<u>&lt;event&gt;</u>, <u>&lt;state&gt;</u>);  <u>&lt;time&gt;</u> ::= number MINOR CYCLES   number SECONDS  <u>&lt;state&gt;</u> ::= ON   OFF   + PULSE   - PULSE</li> <li>• SIGNAL (<u>&lt;event&gt;</u>, {<u>ON</u>   <u>OFF</u>});</li> </ul> <p><b>PROCESS CONTROL FUNCTIONS</b></p> <ul style="list-style-type: none"> <li>• bit function call ::= EREAD(<u>&lt;event&gt;</u>)</li> <li>• bit function call ::= INVOKED(<u>&lt;task name&gt;</u>)</li> <li>• integer function call ::= TIME</li> <li>• integer function call ::= MINOR CYCLE NUMBER</li> </ul>	<p><b>DATA DECLARATIONS</b></p> <ul style="list-style-type: none"> <li>• TASK (<u>&lt;task name&gt;</u>);</li> <li>• EVENT (<u>&lt;event name&gt;</u>);</li> <li>• LOCAL COPY (<u>&lt;data block&gt;</u>, <u>&lt;type&gt;</u>);</li> <li>• GLOBAL COPY (<u>&lt;data block&gt;</u>, <u>&lt;type&gt;</u>);  <u>&lt;type&gt;</u> ::= READ   WRITE   UPDATE   TRIGGER</li> <li>• COMSUB (<u>&lt;comsub name&gt;</u>);</li> </ul> <p><b>I/O CONTROL STATEMENTS</b></p> <ul style="list-style-type: none"> <li>• READ (<u>&lt;COMPOOL block name&gt;</u>);</li> <li>• WRITE (<u>&lt;COMPOOL block name&gt;</u>);</li> <li>• ACCESS (<u>&lt;COMPOOL block name&gt;</u>);</li> <li>• BROADCAST (<u>&lt;COMPOOL block name&gt;</u>);</li> <li>• TRIGGER (<u>&lt;COMPOOL block name&gt;</u>, <u>&lt;time&gt;</u>, <u>&lt;delta time&gt;</u>);</li> </ul>

The Process Control statements describe the real-time structuring and control of the Task hierarchy. The SCHEDULE statement was the most complex of these statements and its execution sets up the environment in which the Task will execute. That is, the SCHEDULE statement upon being executed, describes the priority of the task being scheduled and the conditions, both event and time, under which it will be executed.

The CANCEL and TERMINATE verbs de-scheduled or simply prevented the current execution of a Task, respectfully. The various wait statements allow a task to wait for an event or for a relative or absolute time before it is executed. The SIGNAL statement allows a task to control the setting of a real-time event.

The I/O Control statements allow the reading and writing of the global Compool Blocks. The READ and WRITE statements are used with LOCAL'COPYs while the ACCESS and BROADCAST statements have a parallel usage for GLOBAL'COPYs. The time granularity of the DAIS system is based upon the concept of a major frame and minor cycles within that major frame. DAIS has nominally defined a major frame as one second but this can be modified to any reasonable value. Within the current DAIS system, there are 128 minor cycles per major frame. With respect to task executions, execution time granularity is that of the minor cycle or 7.8125 msec for the current time partitioning. When weapon release or other minor cycle time critical events are to occur, it is desirable to have more precision than this minor cycle time granularity would allow. The TRIGGER statement supports this need and is currently specified to have an accuracy of one millisecond.

While DAIS has chosen this set of executive primitives, it differs from those found in languages such as HAL/S, SPL/I, and PEARL. In each of these languages, the real-time constructs vary, yet each is sufficient to handle real-time process control. The importance of a particular set of such primitives is slight compared to the advantages of having a single standardized set.

### 3.4 Avionics Environment

Central to the DAIS concept is the avionics specific nature of the design. Avionic systems have unique requirements that are not always applicable to ground based systems. Since these requirements are unique to embedded computer systems, it was desirable to provide the applications programmer with a set of functions or subroutines of commonly used type. This prevents duplication of development effort and standardizes the calling sequences for these commonly used, avionics subroutines.

The DAIS Mission Software developed a number of commonly used subroutines (Comsubs) for use by all applications programmers. Table 6 lists the Comsubs currently available on DAIS.

### 3.5 Managerial Control: DAIS Software Standards

One of the important attributes required for the development and control of a large software project is the definition and enforcement of a set of design and programming standards. Under the DAIS Mission Software contract, DAIS Mission Software standards were developed that defined standards, rules and guidelines to be followed by the software engineers in the design, implementation, verification and documentation of the DAIS Mission Software. The Standard was intended to be an evolving document. Beginning from the baseline document, the standards were added to and otherwise revised to reflect the actual experience gained from the DAIS Mission Software development effort. Not only did these standards define the guidelines, but they were also authoritative. Any variance from the standards required the approval of the Chief Programmer who had the technical authority for the project.

The software standards document continued to evolve throughout the mission software development effort. In its final evolution, the standards consisted of five chapters and a set of six appendices. A brief description of the standards is as follows:

1. Introduction
2. Architectural Background - This chapter presents the basic overview information on the DAIS system required for the full understanding of mission software.

TABLE 6. DAIS COMMONLY USED SUBROUTINES (COMSUBS)

Comsub	Definition
<u>Vector/Matrix Comsubs:</u>	
MXM REENTRANT (AM,BM,CM)	Matrix-matrix product
MXV REENTRANT (AM,BV,CV)	Matrix-vector product
TRANPOSE REENTRANT (AM,BM)	Transpose of a matrix
VADD REENTRANT (AV,BV,CV)	Vector addition
VDOT REENTRANT (AV,BV:CS)	Vector dot product
VUPDAT REENTRANT (AV,BC,CV)	Vector update
<u>Special Function Comsubs:</u>	
BINARYSEARCH REENTRANT (ARRAY,VALUE,NUM:OUT)	Binary search on an array
REVERSE REENTRANT (WORD,NUM:DROW)	Reverses the bits of a word
SMOOTH REENTRANT (SMOOTHED VALUE,RAW' VALUE,TAU,DT:SMOOTHED' VALUE)	First order lag smoothing comsub
<u>Formatting Comsubs:</u>	
ALTSETFORMAT REENTRANT (REF'PRES:CHAR)	Formats reference pressure for display
CNTRSTATIONFORMAT REEN- TRANT (STATIONS,CHAR)	Formats stations in two strings
CONVERTLATLONG REENTRANT (RAD,DEG:SEC)	Formats latitude/longitude in degrees and seconds
DEGREEFORMAT REENTRANT (RADIAN:CHAR)	Formats radians into degrees for display
DENORMALIZE REENTRANT (INPUT,SP,OUTPUT)	Denormalizes floating point values
ELEVDFORMAT REENTRANT (ELEV,HGT:CHAR)	Formats elevation/ decision height for display

TABLE 6. (cont.)

Comsub	Definition
<u>Formatting Comsubs (cont.)</u>	
FLOATFORMAT REENTRANT (NUMB,COUNT:CHAR)	Formats a floating point integer for display
FLYTOFORMAT REENTRANT (PT,RAD,DIST:CHAR)	Formats fly-to values for display
FUZINGFORMAT REENTRANT (FUZING:CHAR)	Formats the stores fuzing for display
ILSCHNGFORMAT REENTRANT (FREQ,COURSE:CHAR)	Formats the ILS data for display
INTERVAL FORMAT REENTRANT (INTERVAL:CHAR)	Formats the weapon interval for display
JETPAGEFORMAT REENTRANT (JET'TABLE:CHAR)	Formats the programmed jettison data for display
LATLONGFORMAT REENTRANT (RAD,PTYPE:CHAR)	Formats latitude/longitude for display
LIMIT REENTRANT (VAL1, LOW,HIGH:VAL2)	Limits a value at upper and lower limits
MAGVARFORMAT REENTRANT (RAD:CHAR)	Formats the magnetic variation for display
MODEFORMAT REENTRANT (MODE:CHAR)	Formats the release mode for display
NORMALIZE REENTRANT (INPUT,SF:OUTPUT)	Normalizes a fixed point number
OFFSETFORMAT REENTRANT (ID:CHAR)	Formats offset data for display
QUANIDFORMAT REENTRANT (QUAN,ID:CHAR)	Formats the stores quantity and name for display
STATIONFORMAT REENTRANT (WEAP'STATIONS:STATION' WORD)	Formats the stations for display
TCNCHNGFORMAT REENTRANT (CHAN,XY:CHAR)	Formats the TACAN channel for display
UISENCODE REENTRANT (NUMB,COUNT:CHAR)	Formats an integer for display
UHFCHNGFORMAT REENTRANT (FREQ:CHAR)	Formats the UHF channel/frequency for display
WINDSETFORMAT REENTRANT (RAD,VEL:CHAR)	Formats the windset data for display
<p>Note: 1) All vectors are 3 dimensional.  2) All matrices are 3 by 3.  3) Vectors and Matrices are declared as Tables.  4) The last argument in the parameter list is the "output" argument.</p>	



3. Software Structure - This chapter discusses the structure design requirements for the Applications Software. The structuring is based on Higher Order Software (HOS) principles as applied to the DAIS system.
4. Programming Standards - This chapter contains the standard required for the DAIS Mission Software in general and the Applications Software in particular. It discusses:

- DAIS Language Usage
- Application Software Naming Conventions
- Building Blocks for Application Software
- Tasks and Events: Their Control and Interaction.

It is this chapter, more than any other, that is used as a reference by the engineer and programmer in the writing of JOVIAL routines.

5. Executive Programming Standards - This chapter contains information unique to the Executive development, particularly naming conventions and structuring requirements.

Appendix I. - The required system and project information necessary to actually build a software mission.

Appendix II. - Details of the Executive-Application Interface which serve as an Interface Control Document for the particular implementation.

Appendix III. - The system and project unique information required for the proper functioning and execution of the executive implementations.

Appendix IV. - The documentation standard.

Appendix V. - The Testing and Verification requirements associated with the development of the Mission Software.

Appendix VI. - The current set of DAIS Comsubs.

### 3.5.1 Programming Standards

The objective of the programming standards chapter (Chapter 4) of the software standards is to transform the functionally stated design requirements into a set of specifications in terms of the programming language being used. The specifications are in turn documented using the documentation standards (Appendix IV) as a guide.

Specification proceeds in a top-down manner. Beginning with the Master Controller, the hierarchical control map previously derived from the requirements design was reconsidered in terms of JOVIAL programmable blocks and executive capabilities. Based on mission requirements, (e.g., time responses, on-line computations) the functions to be performed were characterized as periodic or aperiodic processes or sequential (either in-line or 'called') subroutines. Specific details such as algorithms, data structure and name scope were integrated into this framework. At each succeeding hierarchical level, Controller and Calculator modules were designed. The internal structure of a lower level module was not specified until its Controller's input, process and outputs were completely defined.

Sections of the programming standards chapter specify guidelines for the use of JOVIAL and the executive system. This permits the detailed design and implementation of the DAIS Mission Software. The main areas addressed were:

- Naming Conventions
- DAIS Language Usage of J73/I
- DAIS Built-In Functions
- DAIS Comsubs
- Real-Time and Executive Interface

Due to the differences in the nature of Applications Programs versus System Programs, a special chapter on Executive Programming Standards was required. While the basic programming practices hold, differences occur of necessity with respect to:

- Naming Conventions
- Language Use of J73/I

### 3.5.2 Documentation Standards

DAIS Mission Software was designed using a combination of two basic programming styles: structured programming and top-down techniques. The structured programming concept is characterized by a limited, ordered set of program constructs. The use of top-down techniques results in program flow which can be compared to the organization of a book; the "table of contents" specifies the entire program flow on page one and each "chapter" is the expansion of a particular block. Conventional flow chart techniques cannot adequately convey these organizations. The block structure, the scope, and the data flow inherent in any structured top-down program must be represented by a structured flow chart.

The DAIS Mission Software made use of a flowcharting tool, IRATE, that automatically generated structured flowcharts from a single command string. The basic elements of these flowcharts represent J73/I constructs. Ideally, all flowcharts should be computer generated by a single tool for consistency, however, equivalent flowcharts, generated by hand using the same symbols, were acceptable when rapid documentation turnaround was required.

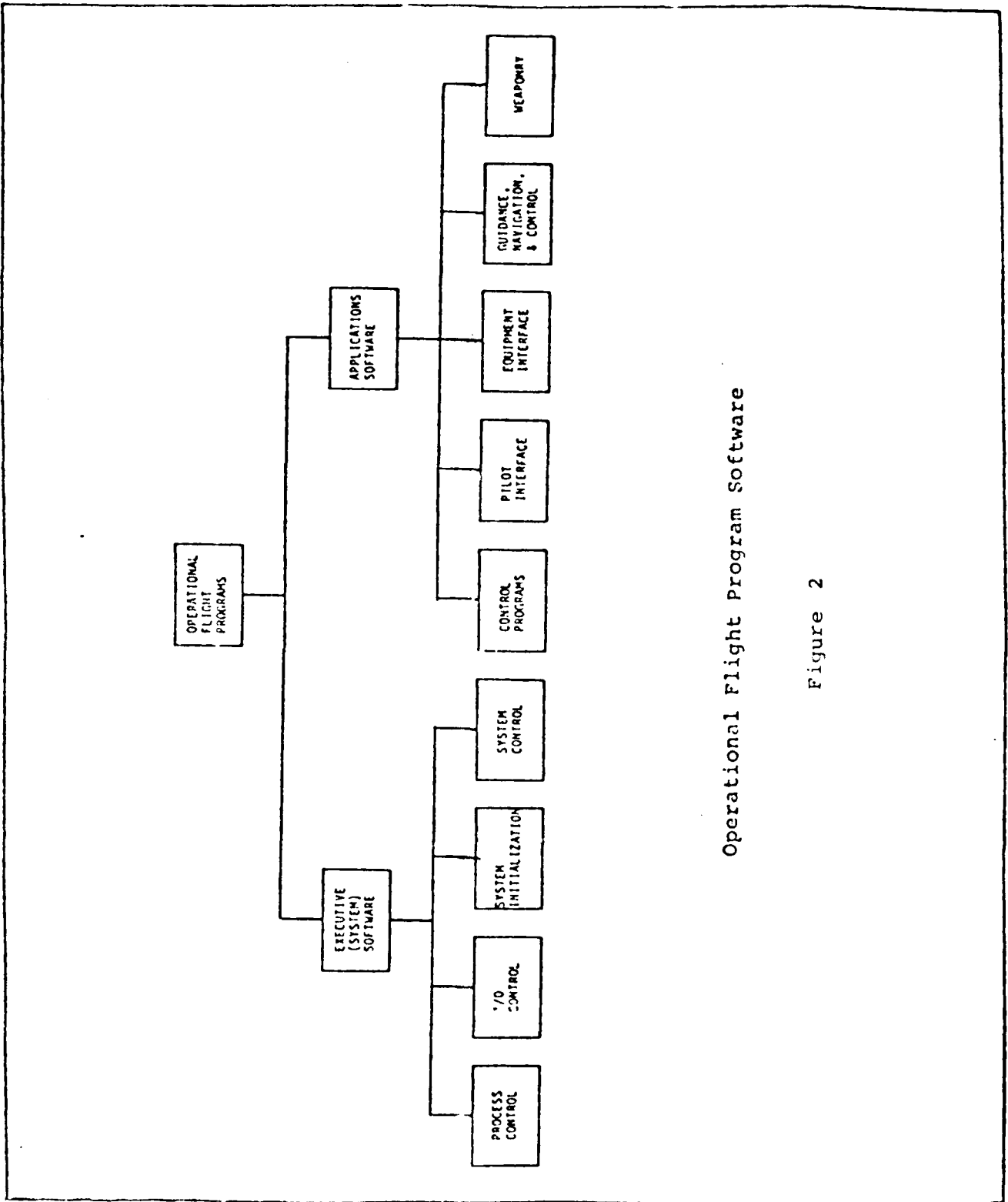
## SECTION IV. TECHNICAL DESIGN

Operational Flight Program (OFP) Software may be broadly divided into two parts: Executive (or Systems) Software, and Applications Software. Figure 2 shows this logical dichotomy along with a lower level of partitioning. It should be noted that there are many possibilities of partitioning the various avionics functions. For example, the Applications Software's Control Programs could be placed under the Executive (System) Software area. The structure shown is one possibility that reflects the OFP used for the DAIS Mission Software.

### 4.1 Executive Software Overview

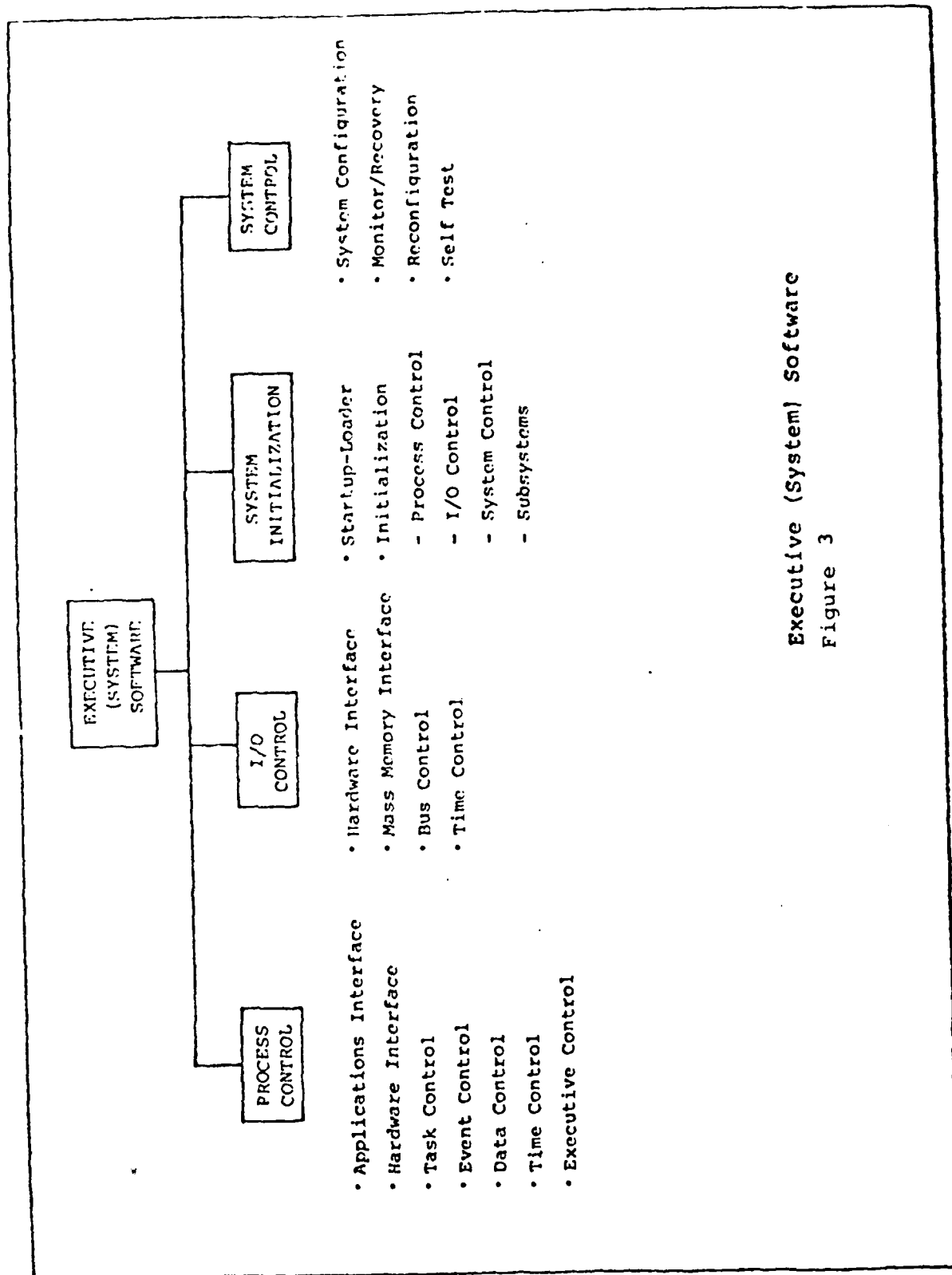
There are characteristic actions that must be performed by avionic and other real-time process control computers. These include the process control of real-time tasks (and their interactions); specialized I/O; system initialization; and system control considerations. Man-rated or process control systems, often closed loop, must continue to function if at all possible in contrast to non-real-time systems which may "crash" without creating hazardous situations. These four areas constitute the Executive Software and are shown in Figure 3 along with a lower level of partitioning.

The development of an executive can be viewed from two different viewpoints. In one view, the executive raises the level of the Applications Software interface. Instead of each Application Program being involved with the details of I/O (e.g., bus control), a logical interface is available to the Applications programmer as a primitive which performs the function desired (e.g., READ). In addition, system-wide considerations such as the Monitor/Recovery function are removed from the concern of each applications program by providing the protection at a "higher" level.



Operational Flight Program Software

Figure 2



Executive (System) Software  
Figure 3

In the other view, an executive is used to optimize system resources. That is, the executive attempts to maximize CPU and I/O subsystems' efficiency. This is accomplished by multiprocessing and by controlling the details of the I/O interactions.

Avionic system designs can also be considered from these two different viewpoints. For example, the principle advantages involved in raising the level of the system interface, from the Applications Software point of view, is to make the Applications Software independent of:

- The processor-memory-I/O network (e.g., single processor, multi-processor, federated or distributive systems).
- Details of I/O implementation (direct I/O or multiplexed bus).
- Partitioning of software across processors.
- Executive implementation (e.g., static task tables or dynamic task tables).

In addition, it is desirable to have:

- Automatic Synchronization and control (e.g., of data conflicts; I/O handling; interprocessor communication).
- Invariant real-time capabilities on different processors or implementations.

In order to obtain these benefits the brunt of the differences between various system designs need be borne by the Executive Software which is processor specific. Not only do processor instruction sets and implementation languages vary, but so do the processor-memory-I/O networks, I/O mechanisms, and reliability requirements. Yet with all these possible variances, the basic functions required by the Applications Software remain constant. It is incumbent on the Executive

Software to support the desired Applications Software interface, while simultaneously optimizing the resources of a given system.

In the current DAIS implementation, the I/O functions of the network are controlled in a master/slave fashion. One processor is designated the Master Processor and any remaining processors are termed Remote Processors. The Master Processor has responsibility for the control and servicing of the data bus including synchronous and asynchronous message traffic control among the system components. The Master Processor thus serves as the Bus Controller for the network.

Given the functional breakdown of the DAIS Executive in Figure 3, the actual Process Control portion of the executive function must be distributed among the processors to correctly implement real-time process interactions. The functional areas within the process control reflect the structural elements being handled; task, events, data, and time; interfaces both to the Applications Programs and to the physical hardware and additionally the control flow through the process controller itself.

The I/O Control portion of the executive is located in the Master Processor, each remote processor containing a simple interrupt handler. The I/O Control functional areas in turn reflect the standard elements being handled: the bus and time; the hardware interface; and any special mass memory interface.

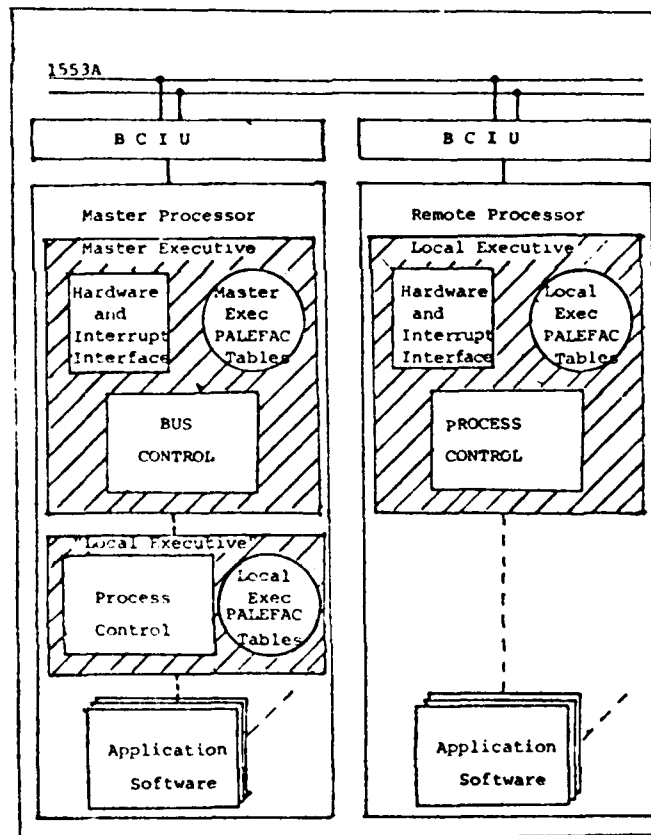
The Process Control and I/O Control functions form the core of the DAIS Executive. The Systems Initialization function is somewhat dependent on the actual system and operating procedures while the Systems Control is concerned with an extra layer of Master/Recovery and Reconfiguration capability.



The DAIS Executive implementation optimizes process and I/O control through the use of executive tables. These tables are statically generated at pre-run time instead of being dynamically linked at run time. This table generation function is performed by the Partitioning, Analyzing and Link-Edit FACility (PALEFAC).

Relating this terminology, the physical and functional breakdown of the DAIS Executive system is shown in Figure 4.

FIGURE 4. EXECUTIVE STRUCTURE BREAKDOWN



#### 4.2 Application Software Overview

The Application Software of an Operation Flight Program (OFP) is responsible for implementing all phases of an aircraft mission. It must compute navigation, weapon delivery and guidance data; it must be able to input data from the avionics sensors and in some cases output control commands to the sensors; it must communicate with the cockpit controls and displays; and it must be able to control the periodic execution of the above functions.

A general partitioning of the Applications Software is shown in Figure 5 along with a lower level of partitioning. Applications Software can be viewed as having three conceptual roles: Control, Equipment Interface, and Avionics.

Control is reflected in Figure 5 as "Control Programs". These programs control the sequencing of the Avionics functions. High level control should not be carried out within algorithmic functions but rather should be implemented as separate control software. This then allows the Avionic functions to be independent of mission phases.

Equipment Interface is reflected in Figure 5 as "Equipment Interface". These programs form a buffer between the sensors, controls and displays and the rest of the Applications Software. Each module in this segment is responsible for converting/formatting data for the equipment since the application software employs floating point while the equipment I/O is in fixed point quantities. Equipment specific testing can also be carried out in these modules.

Avionics is reflected in the Figure 5 as "Pilot Interface"; "Guidance, Navigation and Control"; and "Weaponry". These modules perform algorithmic and bookkeeping functions; only a low level of control is to be found in these routines. Avionics has been further broken into these three functional areas due to their independent characteristics and purposes.

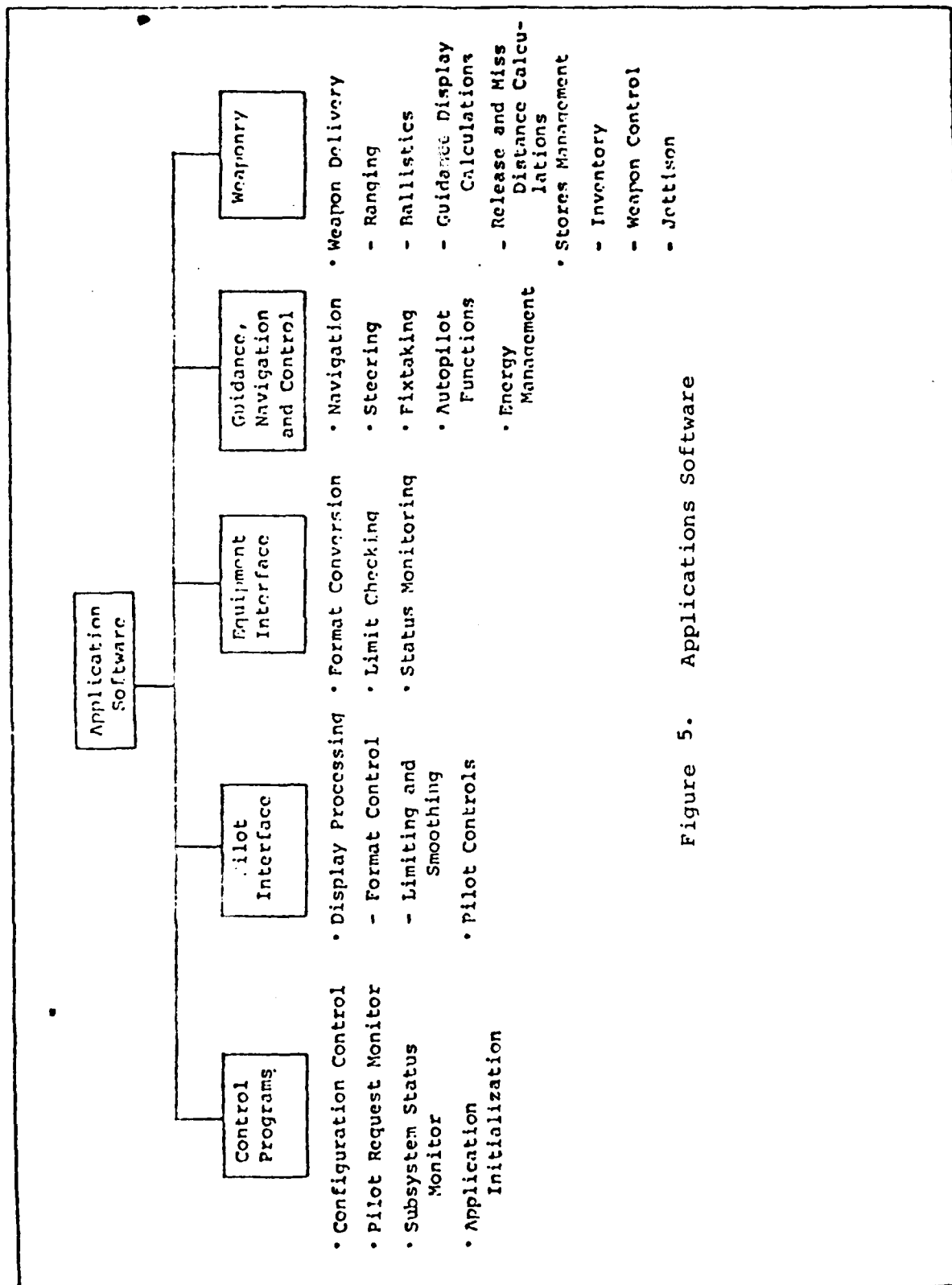


Figure 5. Applications Software

The actual development of OFP Applications Software is highly affected by the range of functions necessary to meet Mission Requirements. While the Mission Requirements usually entail the functions delineated in Figure 5, each of these functions may be quite elaborate with many possible sub-functions or may be so simplified as to be included in another function, e.g., Navigation may become a part of Weapon Delivery. Other factors that affect the design of the Applications Software include the sensor available to perform the avionics functions; the pilot controls and displays used; the interfaces to the sensors, controls, and displays; the executive services available; the subsystems monitored; and the required backup and reconfiguration ability.

Within the framework of meeting specific Mission Requirements, the Application Software should also fulfill several other goals in order to effectively reduce the system's total life cycle cost:

1. Ease of comprehension, generation and documentation - The software should be understandable, easy to write, and easy to document. This will in turn lend to the accomplishment of the other goals.
2. Ease of Modification and Maintenance - The software should be able to easily changed to meet new requirements and interfaces.
3. Ease of Testing - The software should be constructed so that it is easy to test and isolate errors.
4. Portability and Flexibility - The software should be able to be used with different avionics systems and/or different avionics missions.
5. Well-ordered Structure - The Applications Software should consist of (largely) independent modules that are well-ordered relative to each other.

6. Reliability - The software should be highly reliable and should provide graceful degradation of system capabilities in case of component failure, where possible.
7. Execution time and size - The software must be able to meet the real-time constraints of the avionics mission, and the memory size limitations.

One of the goals of the DAIS system was to develop mission software that was useable not just for one OFP, but could be easily modified to meet the needs of several OFPs. The DAIS Applications Software has been developed specifically to meet this goal.

The DAIS system can be thought of as a multilayered system. At the lowest level, as shown in Figure 6, is the hardware. The executive forms a middle layer in interfacing with the hardware and providing a general functional interface to the applications software. At the top level is the applications software, which can itself be further divided into layers. Three levels of applications software are shown in Figure 6.

The lowest applications software level is the equipment interface. This includes both the Equipment Interface software, which interacts with the sensors, and portions of the Pilot Interface software, which interact with the controls and displays. This software effectively isolates the computational portions of applications software from the particular equipment interfaces.

The middle layer of applications software is the computational level. This is the heart of the applications software, where all "real" computations occur. Only localized control is done in this area and no equipment interface control. This layer includes the Navigation, Guidance and Control; Weapon Delivery; and Stores Management Computational Software; and the non-equipment interface portions of the Pilot Interface Software.

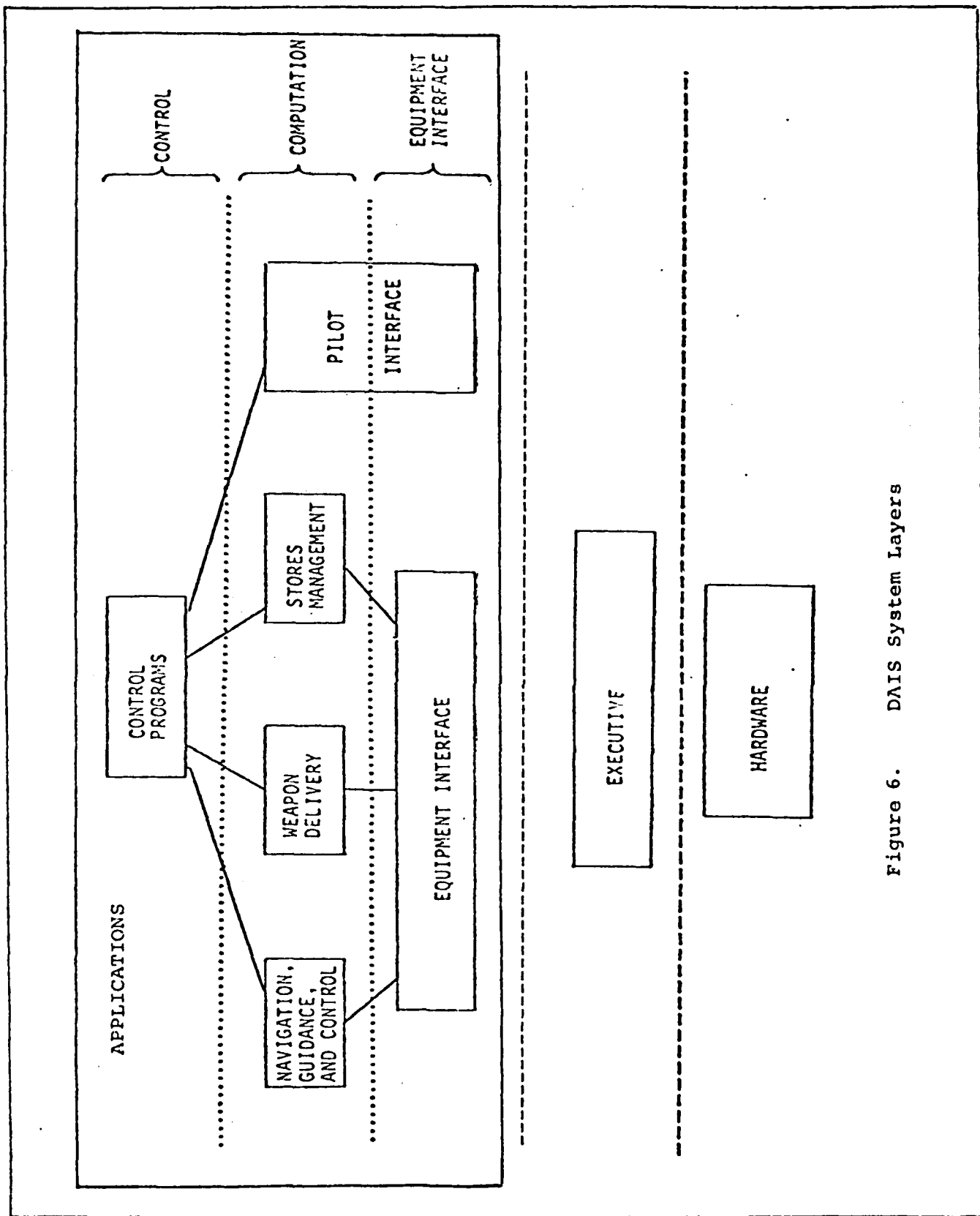


Figure 6. DAIS System Layers

The top layer consists of the Control Programs. This software does not perform calculations per se, but controls which portion of the middle layer is active.

The applications software is isolated from the hardware by the executive. The hardware may change completely without being visible to the applications software. The computational portion of the software is further isolated from the sensors, controls, and displays by the equipment interface. A sensor may change, but only the equipment interface will require modification as long as the sensor is more or less functionally the same. The computational algorithms are also separated from the global control software. Instead of having the control logic sprinkled throughout the software, it is isolated to the extent possible. This leaves the computational modules as mainly algorithmic modules. Changes to the sequencing or control of the modules or changes to equipment interface do not affect the actual algorithmic software.

Changes to the system are thus usually localized to one program or at least one area. Also different types of software are functionally separated. Tested algorithms do not usually need to be retested for changes in the control software or sensor interfaces. The layering thus supports easy modification of the software.

#### 4.3 Application/Executive Interface

Within the life cycle of an avionics system, there usually arises a requirement to upgrade the system with respect to some equipment (e.g., sensors, weapons) or some algorithm (e.g., steering, navigation). These changes in turn dictate modification of the avionics Operational Flight Program (OFP). The first digital avionic processors were programmed in assembly code and were difficult to modify. The code had to be hand crafted in order to obtain all the performance desired. Modification to such OFPs was not only difficult, but the addition of any new feature often required a major redesign

and deletion of current features. The Air Force B-1 and F-16 programs have specified the use of a Higher Order Language (HOL), J3B, for their avionic systems. Similarly, NASA has selected HAL/S for the Space Shuttle. The use of a HOL reduces the classes of errors that are possible, and greatly aids in the understanding of and modification of the resultant OFP in a much more straightforward and less error prone fashion. The use of a HOL has also largely isolated the OFP from the characteristics of the target machines. This separation of the OFP from specific machine characteristics in turn allows for initial code development and module testing on a large host computer. This has the advantages of easier access, earlier development and usually more complete diagnostic tools for the software development than available with the actual target computer.

The DAIS Mission Software operates in a real-time environment. The Software must be able to respond to sensors and control actuators. It must have the ability to execute tasks "periodically" in order to sample data or to execute an algorithm which updates current state information. The capability to monitor the real-time control process itself must also be present. This is required in order to take corrective action upon sensor or actuator failure, or upon failure in the control program. The execution of some missions requires the capability of executing tasks at appropriate times. These forms of task interaction (the initiation of tasks at a time or upon command; the periodic execution of a task; the conditional execution of a task on the occurrence of an event or on the detection of an error) require certain basic executive capabilities. These capabilities are of necessity to be found in real-time process control executives.

The specification of the executive interfaces in JOVIAL is, in effect, a real-time language augmentation to JOVIAL at least from the Applications Software point of view. At the same time it provides a set of "executive primitives" that define the basic operations which the executive must perform upon the tasks.



The description of the various real-time statements and the executive interface given below therefore becomes part of basic programming language used in DAIS Applications Software. The syntax and semantics of the various real-time statements are the allowable "real-time statements" for the DAIS Mission Software. Other process control languages have of necessity similar real-time constructs. Examples of such languages include NASA's HAL/S, the US Navy's SPL/I, and the European PEARL.

The augmented real-time statements are of three forms: Real-Time Declarations, Real-Time Built-In Functions, and Real-Time Statements. The Declarations provide the special data structure required for the DAIS executive system. The Statements are the real-time primitives that interact with the executive. The Built-In Functions allow access to certain real-time information.

#### 4.3.1 Real-Time Declarations

Real-Time Declarations are used to declare the real-time entities referred to within a Task. There are four kinds of Real-Time Declarations:

- Task Declarations
- Event Declarations
- Compool Block Declarations
- Comsub Declarations

##### 4.3.1.1 Task Declarations

Task Declarations are used to declare tasks referred to in Real-Time Statements. They are used to create the appropriate data base required for task manipulation within the DAIS system. The Task declaration has the following form:

```
<task statement> ::= TASK (<task name>);
```

A TASK declaration must appear for each task referenced within the compilation unit.

Examples:           TASK(CP02CONFIG);  
                    TASK(SP21WIND);  
                    TASK(QP16TACANIN);

#### 4.3.1.2 Event Declarations

Event declarations are used to declare Events referenced in the WAIT, SIGNAL and EREAD real-time constructs. They are used to create the appropriate data base required for event manipulation within the DAIS system. The EVENT declaration has the following form:

<event statement> ::= EVENT (<event name>);

The event must be declared whether it is an explicit event provided by the Applications Software programmer or an implicit Compool block update, or implicit Task state event.

Examples:           EVENT(E005IMFKSK);  
                    EVENT(SP04NAVPROP);  
                    EVENT(IC33SROUT);

#### 4.3.1.3 Compool Block Declarations

Compool block declarations are used to declare any Compool Blocks referenced in READ, WRITE, ACCESS, BROADCAST, or TRIGGER statements. There are two types of Compool block declarations.

Local - used to create both the local copy of the compool block and to allow controlled access to the referenced compool. This declaration is used in normal tasks.

Global - used to access the referenced compool block. This declaration can only be used in privileged mode tasks.

The two Compool block declarations have the following form:

```
<local copy statement> ::= LOCAL'COPY (<data block>,
                                     <all types>);
<global copy statement> ::= <GLOBAL'COPY>(<data block>,
                                     <type>);
<all types> ::= <type>|TRIGGER|FORCED'READ
<type> ::= READ|WRITE|UPDATE
```

The type identifies how the Compool block is referenced within the task in which it is declared. TRIGGER means that it is referenced only in TRIGGER statements, and may appear only in LOCAL'COPY statements. READ means that the compool block is referenced only in READ or ACCESS statements. WRITE means that the Compool block is referenced only in WRITE or BROADCAST statements. UPDATE means that the Compool block is referenced in both READ or ACCESS, and WRITE or BROADCAST statements within the task. FORCED'READ means that the Compool block is referenced only in FORCED'READ statements.

Examples:        LOCAL'COPY(NC07DIRCOS,UPDATE);  
                  GLOBAL'COPY(PC60SWITCHES,READ);  
                  GLOBAL'COPY(NC50LASER,WRITE);  
                  LOCAL'COPY(PC50SWITCHES,FORCED'READ);  
                  LOCAL'COPY(SC91RFLEASE,TRIGGER);

Since GLOBAL'COPY statements are restricted to PRIVILEGED' MODE'TASKs their usage is also correspondingly restricted.

#### 4.3.1.4 Comsub Declarations

Comsub declarations are used to declare Comsubs called within the Task. They are used to create the appropriate inter-action with the DAIS system. The Comsub declaration has the following form:

`<comsub statement>::= COMSUB (<comsub name>);`

A Comsub declaration must appear for each Comsub referenced within the compilation unit. It should be noted that the DAIS Built-In functions (e.g., SIN, COS) are not considered as Comsubs but rather as part of the basic language available to the programmer and should, therefore, not be declared.

Examples:           COMSUB(MXV);  
                      COMSUB(VDOT);  
                      COMSUB(TRANPOSE);

#### 4.3.2 Real-Time Statements

The Application Software requests the services of the Executive through Real-Time Statements. There are eleven kinds of Real-Time Statements:

- Schedule Statements
- Cancel Statements
- Terminate Statements
- Wait Statements
- Signal Statements
- Read Statements
- Write Statements
- Trigger Statements
- Access Statements
- Broadcast Statements
- Forced Read Statements.

Real-Time Statements present the Applications Software programmer with the basic primitives to be used in dealing with real-time processes and for interfacing to the executive. They pass appropriate information to the executive as parameters.

#### 4.3.2.1 Schedule Statements

Schedule Statements are used by a Task to Schedule another Task. A Schedule Statement must include the following information:

- The name of the Scheduled Task
- The priority of the Scheduled Task
- The Latched Conditions (if any) in the Event Condition Set of the Task
- The Unlatched Conditions (if any) in the Event Condition Set of the Task
- The period and phase of a Minor Cycle Event (if any) in the Event Condition Set of the Task.

The Schedule statement has the following form:

```
<schedule statement> ::=  
  SCHEDULE (<task name>, PRIORITY=<prio> {<unlatched> } {<latched> }  
                                     μ                μ  
<prio> ::= <integer> | -<integer> | PRIVILEGED  
<unlatched> ::= ,UPON<event expression>  
<latched> ::= ,IF<event expression>  
<time field> ::= ,PHASE=<integer>, PERIOD=<integer>  
<event expression> ::= <factor> | <factor> AND <event expression>  
<factor> ::= [ <factor2> ] | <factor2>  
<factor2> ::= <or set> = ON | <or set> = OFF | <or set>  
<or set> ::= <event> | <event> OR <or set>  
<integer> ::= a legal JOVIAL integer constant  
<event> ::= <ext name>
```

The semantics of the SCHEDULE statement are as follows:

- The <task name> is the name of the task to be invoked.
- The <prio> field indicates the priority of the task to be scheduled. This value may be either an absolute integer or a negatively signed integer, or the word PRIVILEGED. If it is an absolute integer, then this is the "absolute" value of the priority of the task with respect to the DAIS Software system. The smaller the value, the higher the priority. If <prio> is a negative integer, this assigns the priority relative to the scheduling task and establishes the relative priority among all of its siblings. A <prio> of -1 establishes the scheduled task as having a priority immediately lower than that of its parent, the one who scheduled it. A <prio> of -2 is of priority immediately less than that of the sibling task scheduled with a <prio> of -1. If the <prio> is the word PRIVILEGED, the task is scheduled as a privileged mode task.
- Priorities used by the DAIS Mission Software Tasks are of relative priority. Exceptions to this rule require central approval. The use of absolute priorities is limited to time critical tasks, and in general will be associated with the use of the TRIGGER statement.
- The Latched and Unlatched parts of the Condition Sets are defined by their respective <event expression>s. The syntax of the event expression allows the combination of events to be evaluated as either ON or OFF in various combinations with AND and OR.
- The integer associated with PHASE must take a value from 0 to PERIOD-1 indicating the appropriate phase.

- The integer associated with PERIOD must be either 1, 2, 4, 8, 16, 32, 64, or 128. This corresponds to the number of minor cycles that must elapse before the task is scheduled again.

Examples:

```
SCHEDULE(QP33FLRSR,PRIORITY=PRIVILEGED,PHASE=3,PERIOD=8;
SHCHEDULE(QP11DOPOUTCOR,PRIORITY=-15,IF E035DOPON=ON,
    PHASE=1,PERIOD=4);
SCHEDULE(CP05IMFKHAND,PRIORITY=-5,UPON E005IMFKSK=ON);
```

#### 4.3.2.2 Cancel Statements

The Cancel Statement is used by one Task to put another Task into an UNINVOKED state. The Cancel Statement includes the name of the Task to be Cancelled. This Task must either be the Task within which the statement is executed, or a son of that Task. If a son is cancelled, all the dependents of the son are also cancelled automatically. If a Task attempts to Cancel itself, it will not affect its own state, but will Cancel all of its descendants. If a Task specifies itself in a Cancel Statement, it must be declared in a Task Declaration within itself. The Cancel Statement has the following form:

```
<cancel statement> ::= CANCEL (<task name>);
```

Here <task name> is the name of the task to be cancelled.

Examples:           CANCEL(QP02IMSOUTTORQ);  
                      CANCEL(SP40UHFCHG);

#### 4.3.2.3 Terminate Statements

The Terminate Statement functions identically to the Cancel Statement, except that it places the named Task into the INACTIVE state instead of the UNINVOKED state. The Terminate Statement has the following form:

```
<terminate statement>::=  TERMINATE (<task name>);
```

Here <task name> is the name of the task to be terminated.

Examples:           TERMINATE(QP60DEKIN);  
                  TERMINATE(DP41IMFKVAR);

#### 4.3.2.4 Wait Statements

Wait Statements are used by Tasks to place themselves into the WAITING state pending certain event occurrences. There are four forms of Wait Statements in the DAIS system.

- Mission Time Waits
- Relative Time Waits
- Latched Waits, and
- Unlatched Waits.

These various Waits are implemented in the following form:

```
<wait until statement>::=  WAIT'UNTIL (<time>);  
  
<wait for statement>::=   WAIT'FOR (<time>);  
  
<wait statement>::=      WAIT (<event>, <state>);  
  
                  <time>::=   <number> MINOR'CYCLES  
                              | <number> SECONDS  
  
                  <state>::=   ON|OFF|+PULSE|-PULSE  
  
                  <number>::=  a legal JOVIAL unsigned  
                              integer constant
```



The WAIT'UNTIL is a Mission Time Wait and places the Task into the WAITING state until the specified Mission time has occurred. If the specified time has already occurred, the task is not put in a waiting state.

The WAIT'FOR is a Relative Time Wait and places the Task into the WAITING state for a specified period of time. If the specified period is non-positive the task is not put in a waiting state.

It should be noted that in either the Mission or Relative Time cases, the time may be specified in units of either MINOR'CYCLES or SECONDS.

The WAIT statement is used to place the Task into the WAITING state with respect to an event which is named in the <event> field. Whether the statement is Latched or Unlatched depends upon the <state> designated. ON and OFF reflect Latched events, while +PULSE or -PULSE reflect Unlatched events.

A Latched Wait places the Task into the WAITING state until the specified Event reaches the specified "desired value" of ON or OFF. If the Event already has the desired value, the task is not put in a waiting state.

An Unlatched Wait places the Task into the WAITING state until the specified Event reaches the specified "desired value" of ON or OFF. If the Event already has the desired value, the task is not put in a waiting state.

An Unlatched Wait places the Task into the WAITING state until the specified Event receives a +PULSE (i.e., until the event is set or signalled ON) or receives a -PULSE (i.e., until the event is set or signalled OFF). The task is always put in a waiting state.

Examples:            WAIT(SP04NAVPROP,OFF);  
                      WAIT'FOR(2 MINOR'CYCLES);  
                      WAIT'FOR(122 SECONDS0);

#### 4.3.2.5 Signal Statements

A Signal Statement sets a specified Event to a specified value. The form of the Signal Statement is as follows:

```
<signal statement>::= SIGNAL (<event>, { ON  
OFF } );
```

Here <event> is the name of the event to be set to the value ON or OFF as designated.

Examples:            SIGNAL (E005IMFKSK,ON);  
                      SIGNAL (E035DOPON,ON);  
                      SIGNAL (E035DOPON,OFF);

#### 4.3.2.6 Read Statements

A Read Statement copies the value of a specified Compool Block into the corresponding Local Copy. The form of this statement is as follows:

```
<read statement>::= READ (<compool block name>);  
  
                      <compool block name>::= <ext name>
```

Here the <Compool block name> is the name of the Compool block to be read into the corresponding Local Copy.

Examples:            READ (NC07DIRCOS);  
                      READ (CC04REQUEST);

#### 4.3.2.7 Write Statements

A Write Statement copies the corresponding Local Copy into the specified Compool Block. The form of this statement is as follows:

```
<write statement>::= WRITE (<compool block name>);
```

Here the <Compool block name> is the name of the Compool block which is to receive the value of the Local Copy.

Examples:           WRITE(NC07DIRCOS);  
                      WRITE(WC05SOLQ);

#### 4.3.2.8 Trigger Statements

A Trigger Statement requests the Executive to send the Local Copy of the specified Compool Block to the appropriate remote terminal at a specified time. The specified time must be two Minor Cycles and one Major Frame from the time the Trigger Statement is executed. The form of this statement is as follows:

```
<trigger statement> ::= TRIGGER (<compool block name>),  
                           <trigger time> <delta time>;  
  
    <trigger time> ::= <numeric formula>  
    <delta time> ::= <numeric formula>
```

Here the <Compool block name> is the name of the Compool block which is to receive the value of the Local Copy at the designated critical time. Time is specified by two fields, <time> and <delta time>. The <time> is specified in Mission Time and is scaled in units of minor cycles. The <delta time> field is scaled in clock units of, and has a granularity of, one millisecond. This <delta time> field provides the fine granularity for the TRIGGER statement. It will be in effect added to the <time> field by the executive in order to obtain the designated critical time.

For a single occurrence, the <time> field should in itself be sufficient for critical time calculations. When multiple occurrences are to occur spaced relative to each other, the <delta time> field provides for finer time granularity than a minor cycle. TRIGGER may not be used in Privileged Mode.

Example:            TRIGGER(IC91RELEASE,TGO,DELTATIME);

#### 4.3.2.9 Access Statements

The Access Statement is used in Privileged Mode tasks to correspond to the Read Statement in normal mode tasks. Its form is:

<access statement>::= ACCESS (<Compool block name>);

Here the <Compool block name> is the name of the Compool block which is being directly accessed.

Example:            ACCESS(PC23INSDAT);

#### 4.3.2.10 Broadcast Statements

The Broadcast Statement is used in Privileged Mode tasks to correspond to the Write Statement in normal mode tasks. The form of this statement is:

<broadcast statement>::= BROADCAST (<compool block name>);

Here the <Compool block name> is the name of the Compool block which is being broadcast.

Example:            BROADCAST(IC73IMUOUT);

#### 4.3.2.11 Forced Read Statements

The Forced Read Statement is used in Normal Mode tasks to read data from a device, upon demand of the task. There is an inherent time delay in this statement since it must instruct the device to send the data to the specified Com-pool Block. The form of this statement is as follows:

<forced read statement>::= FORCED'READ (<Compool block name>);

#### 4.3.3 Real-Time Built-In Functions

Real-Time Built-In Functions are used by the Application Software to "read" the value of certain real-time entities. These are used by the DAIS Mission Software as if they were Built-In Functions. There are four Real-Time Built-In Functions.

- EREAD
- INVOKED
- TIME
- MINOR'CYCLE'NUMBER

##### 4.3.3.1 Event Read Function

ERead yields the current value of the Event passed as its argument. This Event must have been previously declared in an Event Declaration. The form of this function is as follows:

<bit function call>::= EREAD (<event>)

Here <event> is the name of the event whose value is to be read. The EREAD of an Implicit Task state event will be ON if the Task is in the ACTIVE state and OFF if the Task is in the INACTIVE state.

#### 4.3.3.2 Task Event Read Function

INVOKED is applied to a Task. It yields ON if the Task is INVOKED, OFF if it is not. This Function may only be applied to a Task scheduled within the Task in which the Function is used. The form of this function is as follows:

`<bit function call>::= INVOKED (<task name>)`

Here <task name> is the name of the Task whose INVOKED state is being tested.

#### 4.3.3.3 Time Read Function

TIME reads the Mission time. This is the time since system initialization and is measured in number of Minor Cycles. The form of this function is as follows:

`<integer function call>::= <TIME>`

#### 4.3.3.4 Minor Cycle Read Function

MINOR'CYCLE'NUMBER reads the value of the current minor cycle number. The value read is returned as an integer with a value between 0 and 127. The form of this function is as follows:

`<integer function call>::= MINOR'CYCLE'NUMBER`

#### 4.3.4 Real-Time Directives

Real-Time Directives are compile-time statements which alter or enable/disable the normal mode Real-Time Constructs.

##### 4.3.4.1 Privileged Mode Directive

There is a single mode directive to distinguish normal mode from privileged mode tasks. This directive must be the first statement in a privileged mode task. Its format is:

```
<privileged mode task directive>::=  
PRIVILEGED'MODE'TASK;
```

A <privileged mode task> is a synchronous task that is effectively executed at a very high priority. It is executed in each appropriate minor cycle before all normal mode tasks. It may directly reference the global Compool blocks, and thus can be used only in special situations. Its normal use is for a task communicating with a piece of equipment.

Tasks which are directed to be "privileged mode" must have the following characteristics.

- Synchronous
- Short duration (i.e., appreciably less than a minor cycle in duration)

The PRIVILEGED'MODE'TASK directive is to be used sparingly and only to improve efficiency. While data interlock protection is insured with its usage, the ability to insure invisible transient error recovery has been lessened.

#### 4.3.4.2 Local Override Directive

The Local Override Directive allows Normal Mode tasks to have direct access to the Global Copies of Compool blocks, while still using Local Copy declarations. The use of this directive defeats the protective mechanisms supplied by the Executive when a Task references a Compool block.

<local copy override directive>::=  
LOCAL 'COPY' OVERRIDE

#### 4.4 Executive System

The DAIS Executive isolates the physical aspects of the DAIS federated system from the Application Software. The Executive allows the Application Software to reference time, remote terminals and information in other processors on a logical level. It masks the federated nature of the system, so that Application Software can be written as if it were to execute on a single, virtual machine. Finally, the DAIS Executive controls and optimizes the use of system-wide resources, such as the data bus and mass memory, and provides mechanisms for error recovery.

##### 4.4.1 Introduction

The DAIS Executive System consists of the run-time Executive Software and the pre-run-time Partitioning, Analysis, and LinkEdit FACility (PALEFAC). This section will primarily deal with the run-time Executive Software. PALEFAC is used to generate the necessary executive table information in order to have a properly functioning run-time mission.



The DAIS Executive Software consists of two parts: a Local Executive and a Master Executive. Every processor in the DAIS federated system contains a Local Executive. On the other hand, only one Master Executive is in operation at any given time in any given configuration. The Local Executive controls operations peculiar to a processor, including control of the Application Software within the processor and local participation in the I/O processes. The Master Executive controls system-wide operations, including control of the data bus, of mass memory, and system-wide initialization and error recovery.

The architecture of the DAIS system implies a separation of functional components, the control of one component over another, and a dependence of one component on another. The DAIS system architecture is depicted in Figure 7 showing the separation of hardware and software functions. The Applications Software is functionally isolated from the hardware by the executive software just as the avionics subsystems are isolated from the computers by the Remote Terminals and Data Bus.

DAIS is a real-time system in which the activities of the Applications Software are coordinated with the passage of real-time in the outer world. The minimum granularity of time to which coordination occurs is known as the Minor Cycle. It is possible to specify or determine the time of an action within one Minor Cycle, but not to a fraction of a Minor Cycle. Thus, the I/O interactions, interprocessor interactions, and task interactions may occur, may be known, and may be controlled within the framework of the Minor Cycle time granularity. This timing is a requirement for I/O control, interprocessor coordination and synchronization, and the Local Executive process handling.

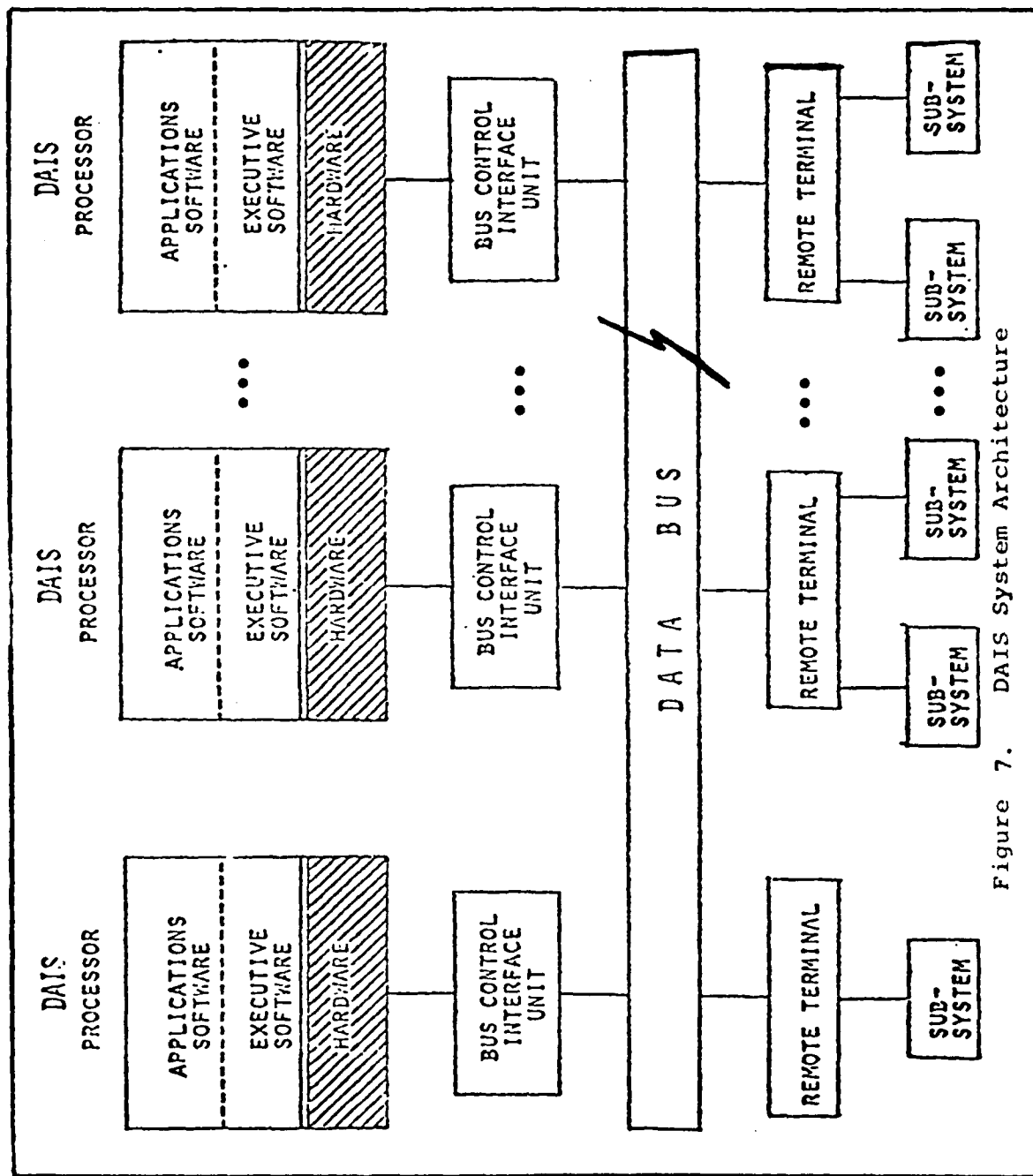


Figure 7. DAIS System Architecture

There is one case in which a finer time granularity control may occur with respect to interaction with the outer world. It is possible to send a sequence of messages to the outer environment with a timing, relative to each other, finer than a Minor Cycle. But this sequence itself cannot be centered better than a Minor Cycle with respect to the outer world.

Because of the multi-processing nature of the DAIS system, a designated active Master Executive within one processor controls the processor configuration. It responds to data bus transmission errors, and controls communication between data bus terminal units. The Local Executive provides the interface between the application functions and the Master Executive (Bus Controller) functions. In addition, there is an interface between the Master Executive and the application functions with respect to system configuration, initialization, and recovery.

The Applications Software controls the execution of software functions by invoking the Executive to schedule and/or activate processes, events, and I/O.

#### 4.4.1.1 Synchronous Action

Actions performed by the DAIS Executive are divided into two classes. One class of actions is performed only in response to a request from one of the components of the DAIS system; these are the asynchronous actions. The other class, synchronous action, is performed periodically.

In order to provide a standard way to specify the periodicity of synchronous actions, time is divided into Major Frames and Minor Cycles. A Major Frame is the longest period of time which may be specified for a synchronous action; a Minor Cycle is the shortest. The number of Minor Cycles to a Major Frame is fixed upon initialization of the DAIS system. This number is an integral power of 2. Within each Major Frame, Minor Cycles are numbered starting with 0.

Synchronous actions include control of the process space of the DAIS processors, and transmission of data between the various components of the DAIS system.

#### 4.4.1.2 Asynchronous Actions

Asynchronous actions are performed upon request by a component of the DAIS system, either software or hardware, and are performed at or near the time when that request is made. The majority of asynchronous activities occurring within the DAIS system do not require the intervention of the Executive. For instance, setting a variable or calling an internal procedure is an asynchronous activity which is performed by the processor hardware. In general, when one DAIS component, either hardware or software, wishes to affect the data space or process state of another component, it is necessary to invoke an asynchronous action on the part of the DAIS Executive.

All asynchronous actions have an inherent latency between the time the request is made and the time the action is performed. This latency depends upon the delays inherent in communications between hardware components, whether within a single processor or via the DAIS Multiplexed Data Bus, and upon the conflicting demands of other asynchronous requests and synchronous actions.

#### 4.4.2 DAIS Executive Functional Description

The DAIS Executive Software is divided into two major functions: the Local Executive, and the Master Executive. In general, the Local Executive controls processes involved with a single processor, while the Master Executive controls processes concerned with the functioning of the system as a whole.

#### 4.4.2.1 Local Executive

Each of the DAIS processors contains a Local Executive. This Local Executive controls the state of the real-time entities existing within its processor, specifically Tasks, Global Copies, and copies of Events (which, like Compool Blocks, may exist in multiple copies, one in each processor in which the Event is referenced).

The Local Executive performs services requested by Tasks in Real-Time Statements. Since a Real-Time Statement executed in one processor may affect the state of a real-time entity in another processor, the Local Executive must be able to send Asynchronous messages requesting services of other processing units (e.g., to Schedule a Task or Update a Compool Block). In addition, the Local Executive must receive such requests from other processors, and must service them properly.

Unlike Asynchronous processes, Synchronous processes are basically under the control of the Master Executive since synchronization is a process involving all processors. However, the Local Executive must also participate in Synchronous processes by signalling Minor Cycle Events and preparing for the reception and transmission of Synchronous Compool Block Update Messages.

Finally, the Local Executive must be capable of starting its processor, and of recognizing and processing errors that may arise.

#### 4.4.2.2 Master Executive

The Master Executive controls communication between the separate processors and remote terminals of the system. This communication exists only in the form of messages which can be sent across the bus. Thus, one major function of the Master Executive is Bus Control.

Each Remote Terminal or processor can request to send Asynchronous messages. There are also Synchronous messages which must always be sent at a given period and phase of a Major Frame. There are also Critically Timed messages which are sent at a specified Mission Time. In addition, the Master Executive has its own messages which are used to check on the correct functioning of the various system components.

The primary function of the Master Executive is to control the sending of these messages. The secondary function is to take corrective action when one of these messages is or appears to be incorrect. The corrective action taken may be of a very simple type; under carefully controlled conditions the message is resent. If this fails, the error cause is assumed to be hardware which has ceased to function properly. Either a logical path around the non-functioning hardware must be found, or the scope of the mission must be changed. This is known as System Configuration Management. System Configuration Management keeps track of the status of all processor and Bus-related hardware used during the mission, and determines the action to be performed when a hardware element fails.

The Master Executive as discussed above is a set of functions which exist in one processor. This processor is called the Master Processor and the BCIU attached to that processor is called the Master BCIU.

To allow for the possibility that the Master Processor or the Master BCIU may fail completely, a second Master Executive exists in another processor. This second Master Executive is called the Monitor. The Master Executive must periodically send a message to the Monitor which informs the Monitor that the Master Executive is still functioning. If the Monitor does not receive this message, it switches to Master Executive mode and takes control of the system. This is called Backup. Recovery exists when the Master Executive detects that the Monitor is not functioning.

Backup may also occur on the failure of a remote processor. If a processor other than the Master or Monitor fails, the Master may cease operation, thus allowing the Monitor to take control of the system. An important aspect of Backup is that the system is now performing a more limited mission. Once Backup occurs, only certain specified critical functions may be performed.

Once Backup has occurred, the pilot has an option to reload the system using fewer than the original number of processors. This is called Reconfiguration. Reconfiguration is started by pilot initiation through the Processor Control Panel (PCP).

A third part of the Master Executive is the Startup/Loader. The function of the Startup/Loader is to load the mission software from Mass Memory into all processors within the system configuration. The Startup/Loader receives control from a Read Only Memory (ROM) Loader.

The ROM Loader gains control of its processor upon power up. It examines the bus and, if not active, assumes control. It then polls all processors in the initial configuration to determine a valid load. The ROM Loader then supervises the loading of the Master Executive from Mass Memory into the Master processor, which by default will have the lowest bus address of all processors. Upon completion of the loading, the ROM Loader then hands control of the system to the Startup/Loader.

If, when the ROM Loader comes up, the bus is active, it goes into an idle loop until one of the following happens:

- commands are received from another ROM Loader (i.e., load a Master Executive)
- commands are received from the Startup/Loader (i.e., load this processor as a Remote)
- a predetermined time has passed, in which case the ROM Loader retries the bus.

### 4.4.3 Interfaces with Real-Time Software

#### 4.4.3.1 General Overview

The DAIS Executive has two principle functions: to provide services to the Applications Software, and to control system-wide functions, such as initialization and recovery. The necessary tasking and I/O interactions of the Applications Software form the functional requirements for the Local Executive Services. The DAIS System Control Procedures form the functional requirements with respect to system-wide control.

The DAIS Applications Software is composed of Tasks, Comsubs, Compool blocks, and Events. Tasks and Comsubs are processing modules, containing executable code and local data. Compool blocks are data modules used for communication between separate Tasks, and between Tasks and the outer world. Events are boolean values used to control the process state of Tasks. Real-Time Statements and Real-Time Built-In Functions are used by Tasks to control and reference the state of other Tasks and the values of Events and Compool blocks.

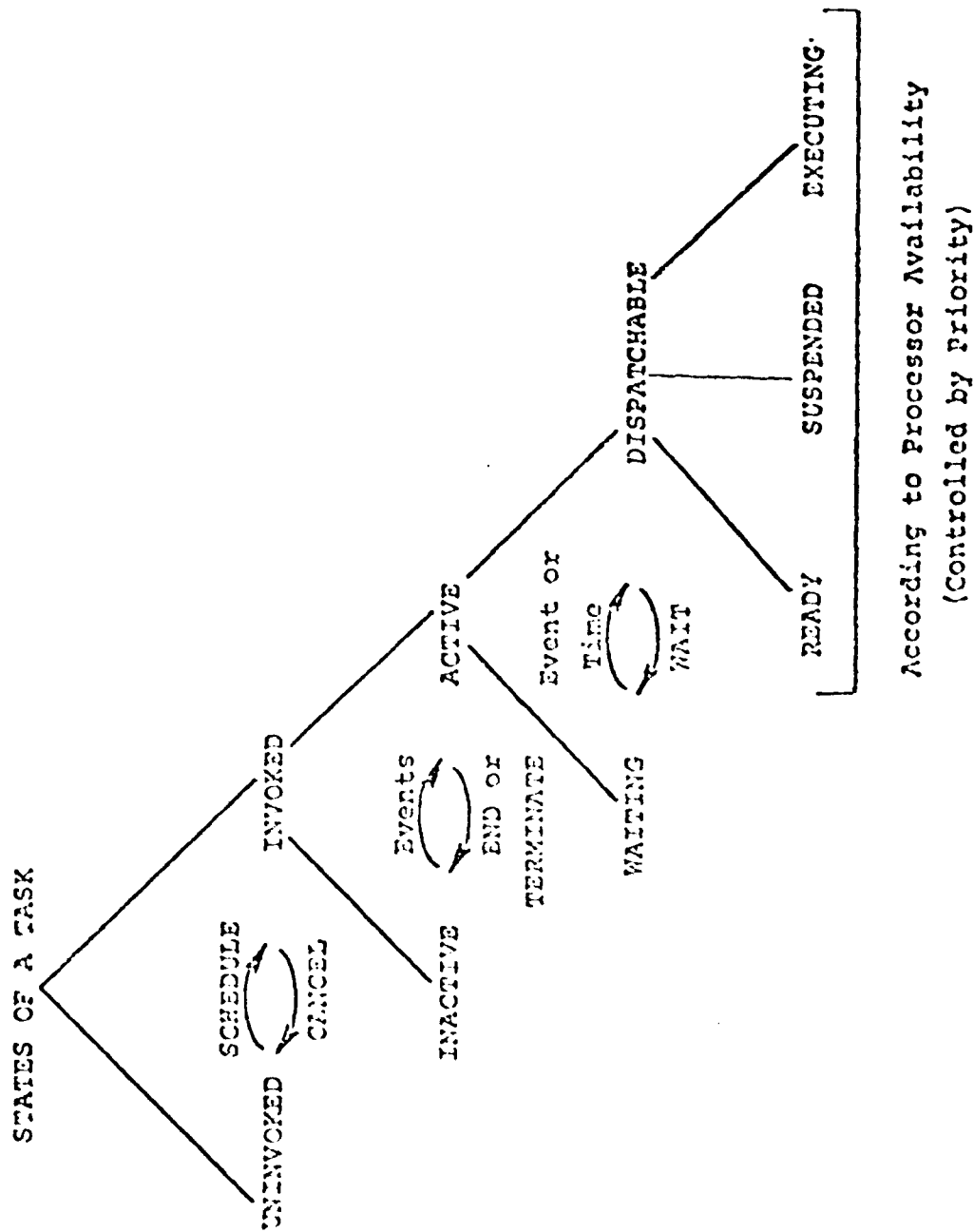
#### 4.4.3.2 Tasks

Tasks are processing modules which can be controlled and executed independently.

4.4.3.2.1 Task States - In order to understand the interactions of Tasks, it is necessary to understand the possible "states" that these dynamic processes may have. At any given instant, each Task in the DAIS Mission Software system has one of the states as shown in Figure 8. It should be noted that not all states are mutually exclusive; in Figure 8 the tree structure shows a subsetting relationship with respect to the various states. Thus,



Figure 8. Task States and Control



INACTIVE and ACTIVE are both substates of INVOKED, and hence, a Task which is INACTIVE (or ACTIVE) is simultaneously also in the INVOKED State. Similarly, a task which is SUSPENDED would also be INVOKED, ACTIVE, and DISPATCHABLE.

Figure 8 also indicates the method of transition from one state to another. For example, a SCHEDULE statement will put an UNINVOKED task into an INVOKED state, while a CANCEL statement will put an INVOKED task into an UNINVOKED state. The meaning of each of these Task states and the means of transition between them as shown in Figure 9 is now discussed in detail.

4.4.3.2.1.1 INVOKED/UNINVOKED - Immediately following system initialization, one Task, the Master Sequencer, is INVOKED by the Executive, while all other Tasks remain in the UNINVOKED state. Thereafter, Tasks can be put into the INVOKED state by a SCHEDULE statement or put into the UNINVOKED state by a CANCEL statement executed within other Tasks.

4.4.3.2.1.2 ACTIVE/INACTIVE - After a task has been SCHEDULED and thus made INVOKED, it is in the INACTIVE substate; however, it has the potential to become ACTIVE, depending upon its Event Condition Set. The Event Condition Set is a collection of Conditions, each of which may be either "ON" or "OFF". Each Condition has a "desired" value. When all the conditions in the Event Condition Set have their desired values, and if the Task is INACTIVE, the Executive will put it into the ACTIVE state. It is possible for a Task to have a null Event Condition Set, in which case it becomes ACTIVE immediately upon becoming INVOKED.

A Task may return from ACTIVE to INACTIVE state from either of two causes: because it completes execution, or because it is forcibly TERMINATED by another Task. In either case, immediately after it returns to the INACTIVE state, the Event Condition Set is evaluated. If all of the

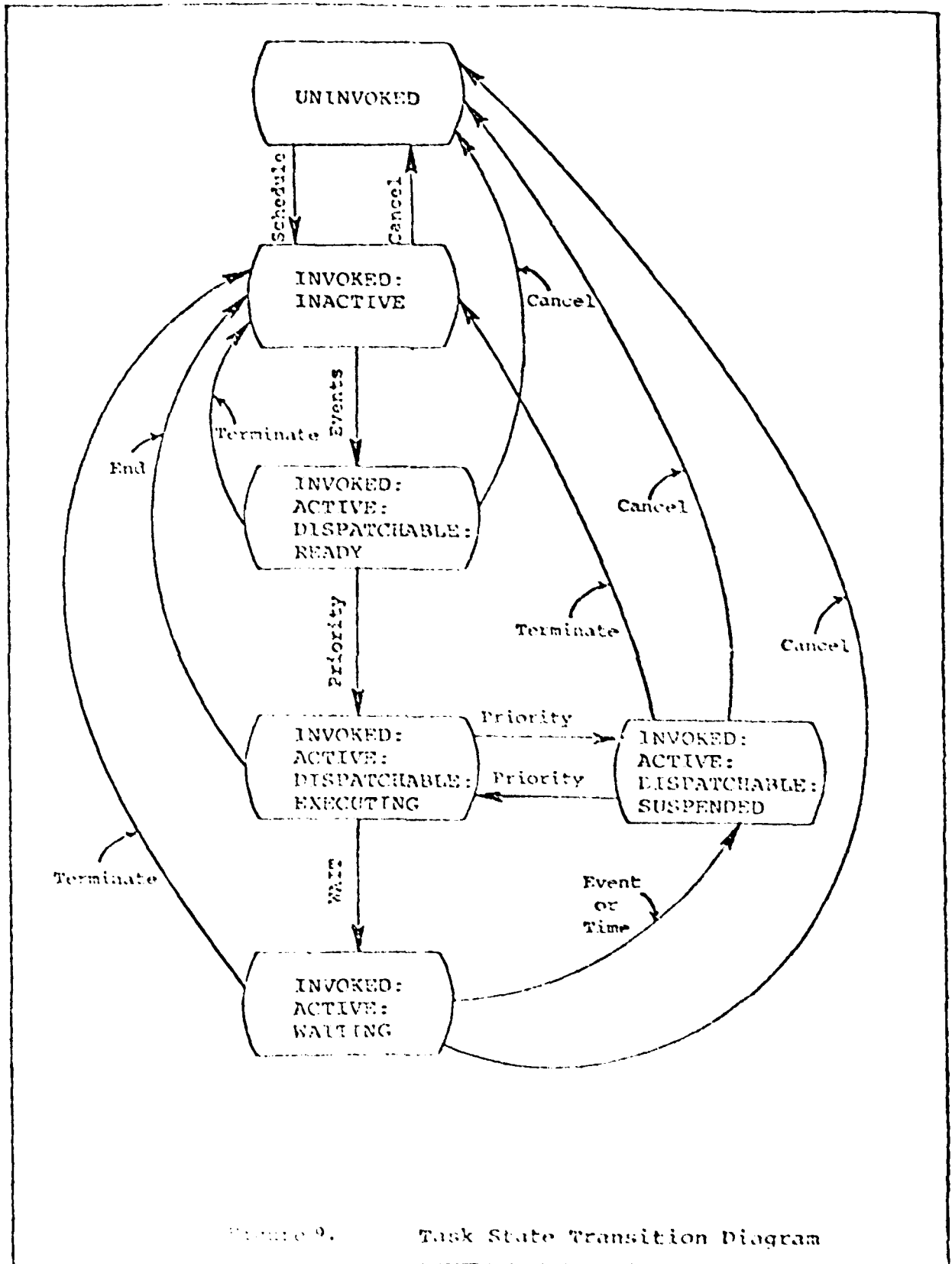


Figure 9. Task State Transition Diagram

Event Conditions have their desired values, then the Task is immediately put back into the ACTIVE state.

4.4.3.2.1.3 WAITING/DISPATCHABLE - When a Task is ACTIVATED, it is also immediately put into the DISPATCHABLE state. If, at any point during its execution, a Task executes a WAIT Statement, specifying either a desired value for an event or a time, the Executive will place the Task into a WAITING state until the condition is satisfied. When this WAIT condition is satisfied, the Task will become DISPATCHABLE.

4.4.3.2.1.4 READY/SUSPENDED/EXECUTING - All DISPATCHABLE Tasks are capable of being executed and should theoretically be executed at any instant within a single DAIS Processor. Tasks are therefore further ordered by Priority in order to resolve these possible conflicts. Whenever the Executive passes control to the Application Software, the highest Priority DISPATCHABLE Task is selected and executed. Since it is not in general possible to immediately execute all DISPATCHABLE Tasks, this state must be further differentiated. There are three DISPATCHABLE substates. If a Task is ACTIVE but has not yet been executed, it is said to be READY. If it has been in the process of execution, but has been interrupted by a higher priority Task, it is said to be SUSPENDED. If it is executing, it is said to be EXECUTING.

4.4.3.2.2 Task Hierarchy - Any Task may be SCHEDULED by exactly one Task, which is then called its Controller. All of the Tasks SCHEDULED by a single Task are said to be its "sons". If a Task has no sons, it is said to have no "descendents"; otherwise, its "descendents" are sons and all of the descendents of its sons. The relationships of Controller and son define the Task Hierarchy.

4.4.3.2.3 Priorities - At any time, there may be many processes potentially executable within any DAIS processor. These include Tasks, Executive actions invoked by Tasks within the processor, and Executive actions invoked by remote terminals or other processors. In order to resolve conflicting demands on the processor, a system of priorities has been adopted. Tasks are divided into two classes: Normal Mode Tasks and Privileged Mode Tasks. As a class, Privileged Mode Tasks and Executive actions have a higher priority than Normal Mode Tasks. Within each class, conflicting demands on the CPU are resolved as described below.

4.4.3.2.3.1 Normal Mode Tasks - Normal Mode Tasks are linearly ordered by priority. At any time, if no Executive actions are called for and if no Privileged Mode Tasks are Active, the CPU will execute the highest priority Active Normal Mode Task.

If during the execution of a Normal Mode Task, an Executive action is called for or if a Privileged Mode Task or a Normal Mode Task with higher priority becomes Active, the original Task is immediately placed into a Suspended state.

4.4.3.2.3.2 Privileged Mode Tasks - Privileged Mode Tasks are not ordered by relative priority; instead, they are executed on a first come, first served basis. When a Privileged Mode Task becomes Active, it is executed immediately. Once a Privileged Mode Task is in Executing State, it is in control of the processor. It can be suspended only when it invokes an Executive action by means of a Real-Time Statement. Upon completion of the action, control is returned to the Task.

Because a Privileged Mode Task cannot be interrupted by an Executive action requested outside of itself, the Compool blocks which it references will not be altered during its execution, unless they are modified by the Data Bus or by the Task itself. Since only Synchronous Compool Blocks can be modified directly by the Data Bus, and then only during

specified Minor Cycles, it is possible to "protect" a Privileged Mode Task against the Data Bus by scheduling it on Minor Cycles during which the Synchronous Compool Blocks which it accesses are not accessed by the Data Bus. Therefore, a Privileged Mode Task is allowed to bypass the protective mechanisms which a Normal Mode Task must use when referencing Compool Blocks.

#### 4.4.3.3 Comsubs

In addition, to Tasks, the DAIS Application Software may contain processing modules known as Comsubs. A Comsub is a computational module which may be called by one or more Tasks and/or Comsubs. A Comsub may communicate with the outside world only through passed parameters.

When a Task calls a Comsub, it is considered to be executing within the code of that Comsub. Thus, it is possible for one Task to be Suspended within the code of a Comsub at the same time that another Task is Executing within the same Comsub. In short, Comsubs are re-entrant.

If two Tasks in different processors call the same Comsub, the Comsub is duplicated in both processors.

#### 4.4.3.4 Compool Blocks

A Compool block may be referenced by Tasks in more than one processor and also, possibly, by a remote terminal. It may therefore be necessary to transmit Compool blocks across the Data Bus. In order to minimize such use of the Data Bus, each processor which uses the values of a Compool block contains a Global Copy of that block. In addition, any remote terminal which sets or uses the Compool block is considered to have a Virtual Copy of the block.

Normal Mode Tasks are not allowed to reference the values within a Global Copy directly. This restriction prevents one Task from referencing data which has been partially updated by another Task. However, Privileged Mode Tasks are allowed

to reference individual values within Global Copies directly, since they cannot be interrupted by another Task or by an Executive action. No other processor or remote terminal is able to set a Global Copy while a Privileged Mode Task is using it, or to use a Global Copy while a Privileged Mode Task is setting it.

4.4.3.4.1 Local Copies - A Normal Mode Task is allowed to reference a Compool block only through the use of a Local Copy. A Local Copy is a data aggregate internal to the Task which uses it. A Local Copy has exactly the same length and internal organization as the Compool Block with which it is associated. The only interfaces Normal Mode Tasks may have with Compool Blocks are:

- They may READ the Compool block into their Local Copy
- They may WRITE from their Local Copy to the Compool block
- They may TRIGGER the Local Copy
- They may READ'DEVICE the Local Copy
- They may WRITE'DEVICE the Local Copy.

READING and WRITEing move the contents of the Local Copy in their entirety to and from the Compool block. TRIGGER is a special statement used only for critically timed Compool blocks.

Since READ, WRITE and TRIGGER invoke Executive actions that operate in Privileged Mode, the use of these statements guarantees that a Normal Mode Task will never reference partially updated data. After it has been READ, or before it has been WRITTEN, the data within a Local Copy may be used and set indiscriminately, without concern for affecting the value of the Compool Block.

4.4.3.4.2 Categories of Compool Blocks - Compool blocks are divided into three categories: Input, Output, and Intertask. Input Compool Blocks are used to input data from remote terminals which may then be used by Tasks. Output Compool Blocks are set by Tasks, and their values are output to remote terminals. Intertask Compool blocks are used for communication between Tasks.

Since a Compool Block may have multiple Global Copies, each in a different processor and possibly a Virtual Copy in a remote terminal, it is necessary to send Compool Update Message(s) across the Data Bus to maintain consistency between the various copies. Compool blocks are further classified as Synchronous, Asynchronous, and Critically Timed.

The various categories of Compool Blocks, and the ways in which they may be referenced by Tasks, are shown in Table 7.

4.4.3.4.2.1 Synchronous Compool Blocks - All Global and Virtual Copies of a Synchronous Compool Block are updated from a single, authoritative copy, either a Global Copy in a processor or a Virtual Copy in a remote terminal. The Update Messages are sent at a specified period and phase. Note that these Update Messages are sent in a pre-determined sequence by the Master Executive, and are invisible to the processor receiving it. Therefore, it is important that Privileged Mode Tasks which reference Global Copies of the Compool block do not run during the Minor Cycles when the Update Messages are being sent, since it is impossible to guarantee data integrity during these Cycles.

When a Synchronous Compool Block is WRITTEN within a Task, the data is simply moved from the Local Copy to the Global Copy within the same processor; the other Global Copies or Virtual Copy are updated from the Copy periodically, as described above.



	SYNCHRONOUS	ASYNCHRONOUS	CRITICALLY TIMED
INPUT	May be used in many Tasks.	May be used in one Task.	
OUTPUT	May be set in one Task.	May be set in many Tasks.	May be triggered in many Tasks.
INTERTASK	May be set in one Task, used in many Tasks.	May be set in many Tasks, used in many Tasks.	

Table 7. Categories of Compool Blocks

4.4.3.4.2.2 Asynchronous Compool BLOCKs - All Global and Virtual Copies of an Asynchronous Compool Block are updated when any one of those copies is updated, either by a WRITE statement or a BROADCAST statement, within a Task, or upon request by a remote terminal. Note that a processor containing only Tasks which WRITE, but do not READ, an Asynchronous Compool Block need not contain a Global Copy of the block.

If a Compool block is referenced only by Tasks within a single processor, the distinction between Synchronous and Asynchronous is immaterial; however, it is preserved for the sake of consistency.

4.4.3.4.2.3 Critically Timed Compool Blocks - Critically Timed Compool Blocks are a special category used only for Output. They are sent to their associated remote terminals at a time specified in a TRIGGER statement.

More precisely, the data in a Local Copy is sent by a TRIGGER statement to a Global Copy in the Master Processor, where it is held until the precise time specified in the TRIGGER statement, at which time it is sent to the associated remote terminal.

4.4.3.4.3 Minor Cycle Tag Words - The first word of every Global Copy of a Compool block is a Minor Cycle Tag Word, which indicates the Minor Cycle during which the Global Copy was last updated. When a Global Copy is updated by the BCIU, this Tag Word is created by the BCIU. When a Global Copy is WRITTEN by a Task, the Tag Word is created by the Executive.

#### 4.4.3.5 Events

Events are used for control between tasks and between a task and the environment. An event has but two possible values: ON and OFF. Events may often, however, be treated in two different fashions. That is, a single event may be

considered to be either a latched event or an unlatched event. To consider an event as latched is simply to consider its state as ON or OFF. To consider an event as unlatched is to consider the pulse, or signal, sent to the event. That is, to consider whether it is, at a given instant, being set ON or OFF (equivalently whether it is sent a positive pulse or a negative pulse). A given event can be considered by one Task as a latched event while simultaneously be considered by another Task as an unlatched event.

4.4.3.5.1 Event Types and Usages - Table 8 shows the various types of Events in the DAIS Mission Software system. It also indicates the statements in which these events may be referenced within the DAIS Mission Software system. There are two broad types of events: Explicit Events and Implicit Events. Explicit Events are named and used by the Applications Software programmer. They contain the meaning which the programmer gives them. Implicit Events are associated with some system meaning and must use the appropriate naming conventions or mechanisms in order to access this system information. In particular, Implicit Events are either associated with a Minor Cycle (via the SCHEDULE statements PHASE=, PERIOD= clause), a Task state, or the updating of a Compool block.

In Table 8 the usage and meaning of each of these event types with respect to the Real-Time statements is indicated. Often a particular event type may not be meaningfully used within some Real-Time statement, and thus it is prohibited in that usage. While the table segregates the Explicit Events into latched and unlatched cases, it should be noted that the Implicit Task state and the Implicit Compool block events can also be treated in these two fashions, but not always meaningfully.

Table 8. Events: Types and Their Usage

Event Type	Real Time Construct	DECLARATIONS EVENT	REAL TIME STATEMENTS				
			SCHEDULE	WAIT	SIGNAL	EREAD	INVOKED
Explicit Events	Treated as Latched	If the Event is used within the compilation unit in either a: WAIT, SIGNAL, or EREAD statement, then it must be declared via an EVENT declaration.	SCHEDULES via: IF clause	WAITS for: ON/OFF	Signal is: ON/OFF <sup>1</sup>	Returns state: ON/OFF	X
	Treated as Unlatched		SCHEDULES via: UPON clause	WAITS for: +PULSE/-PULSE	Signal is: ON/OFF <sup>1</sup>	X	X
Implicit Events	Minor Cycle <sup>2</sup> (via PHASE <sup>2</sup> , PERIOD <sup>2</sup> )		SCHEDULES via: PHASE <sup>2</sup> , PERIOD <sup>2</sup>	X	X	X	X
	TASK state <sup>3</sup>		X	(Corresponds to ACTIVITY state) WAITS for: +PULSE/-PULSE or ON/OFF <sup>4</sup>	X	Returns ACTIVITY state: ON/OFF	Returns INVOKED state: ON/OFF
	COMPOOL block <sup>5</sup>		(Corresponds to UPDATE state) SCHEDULES via: UPON <sup>6</sup>	(Corresponds to UPDATE state) WAITS for: +PULSE/-PULSE <sup>5</sup>	X	X	X

X - Either non-existent, meaningless, or error-prone. In all of these cases it is prohibited from use by the DAIS Mission Software.

1 - In setting an Event state by using the SIGNAL statement, the effect is both to send a pulse and to set the state. Thus, SIGNAL(EOSINHTASK,ON); will set the state of EOSINHTASK to ON, while simultaneously having the effect of a +PULSE.

2 - This is considered to be an Event because of historical reasons. For documentation purposes this will continue to be called an event.

3 - The TASK state event is identified by the use of the TASK name. Thus if QP12DOPIN, an equipment task, is to have its corresponding Event state name referenced, the same name, QP12DOPIN, is used.

4 - The ACTIVITY state may be treated either as a latched or an unlatched event. As a latched event, it provides the information as to whether the named Task is ACTIVE or INACTIVE. Treated as an unlatched event, it can be used to determine when a Task is made ACTIVE or INACTIVE.

5 - The Compool block event is identified by the use of the Compool block name. Thus, if IC12DOPIN, an I/O COMPOOL block, is to have its corresponding update event referenced, the same name, IC12DOPIN, is used.

6 - The Compool block event is set by the DAIS executive system whenever the corresponding Compool block is updated. Thus, the state of the event could be considered to always be ON (after the COMPOOL block's first usage). Thus, it is only the unlatched event aspect that should ever be used.

Explicit Events are under the Applications Software programmer's control. Minor Cycle events are under systems control and only appear in the SCHEDULE statement via the PHASE-, PERIOD= clause. Task state events have an associated Task Activation Event. The Task state event is set ON when the Task is ACTIVATED and set OFF when the Task returns to the INACTIVE or UNINVOKED state. The Activation Event associated with a Task has the same name as the Task. When a Task State event is used by the INVOKED function, the information returned is with respect to the INVOKED state of the task and not the ACTIVE state.

Compool Block Events are set ON when the Compool Block is updated, either by a Task or a remote terminal. The Update Event associated with a Compool block has the same name as the Compool block.

4.4.3.5.1.1 EVENT - An event name must be declared via an EVENT declaration if it is used in either a WAIT, SIGNAL or EREAD statement. If the event name only appears in a SCHEDULE and/or INVOKED statement, it should not be declared via an EVENT statement. This usage of the EVENT declaration is strictly followed for the DAIS Mission Software.

4.4.3.5.1.2 SCHEDULE - All forms of events may appear within a SCHEDULE statement. The differentiation between latched and unlatched occurs through the appropriate use of the IF (latched) and UPON (unlatched) clauses of the SCHEDULE statement. The event names used in these clauses could be either Explicit event names or the Implicit Task state or Implicit Compool block event names.

It should be noted that the use of the Implicit Task statement event in all of the Real-Time Statements is associated with the ACTIVE state of the Task. The one exception is the use of the Real-Time INVOKED function which will test for the INVOKED state of the Task named.

It is only in SCHEDULE statements that Minor Cycle events may occur. However, even here they do not appear directly, but rather via the use of the PHASE=,PERIOD= clause.

It does not make sense to treat an Implicit Compool block event as a latched event since it will be ON. Rather, it is of interest to treat it as unlatched; a pulse is sent when the Compool block is updated although the state is already ON.

4.4.3.5.1.3 WAIT - The WAIT statement may be either of latched or unlatched form, and treat either Explicit events, or the Implicit Task state, or Implicit Compool block events.

4.4.3.5.1.4 SIGNAL - Only Explicit events may be SIGNALLed by the Applications Software. It is prohibited to SIGNAL any Implicit event. Implicit events reflect the systems state and are not under direct Applications Software control.

It should be noted that the differentiation between latched events and unlatched events does not have any meaning for the SIGNAL statement. By setting the state of an event to ON, it both makes the state ON and is a positive pulse. Similarly, setting the state of an event to OFF both makes the state OFF and is a negative pulse.

4.4.3.5.1.5 EREAD - The use of the EREAD function allows access to the state of an event. EREAD will return the value of ON or OFF. Explicit events and Implicit Task state events may be EREAD.

It should be noted that an EREAD of a Compool block is meaningless since its value is always ON.

Since EREAD returns the state of the event, it does not have meaning with respect to unlatched events.

4.2.3.5.1.6 INVOKED - An INVOKED function applies only to Implicit Task state events. It will return the information as to whether the named Task is INVOKED (i.e., ON) or UNINVOKED (i.e., OFF).

Any INVOKED function does not require an EVENT declaration because it is accessing the TASK state event and not the Implicit Activation event.

4.2.3.5.2 Event Condition Sets - Associated with each SCHEDULE statement is its Event Condition Set. Heuristically, when the set of event conditions is met, the Task will become ACTIVE. There may be up to 16 different Conditions in an Event Condition Set, where the time field (PHASE/PERIOD) counts as one Condition.

Within an <or set> there can be an unlimited number of events "or"ed together. This arises from the fact that each of the 16 Conditions of the Event Condition Set is assigned a single bit in the Task Tables. There are 16 such bits for each task. This also indicates the problem with the <or set>. All events of a given <or set> map to the same bit in the Task Table. Thus, the <or set> expression is not a true "or" of all of the events "or"ed, together. Instead, the <or set> is the "last state" change within the <or set> of events. If the <or set> were:

A or B = ON

then it would be satisfied if either A or B were signalled ON. However, if both A and B were currently ON, and say B were signalled OFF, then the <or set> would become unsatisfied since it reports the last state of any of the events within its <or set>. B was the last to change, and this change was to OFF, thus not fulfilling the required condition.

With these limitations in mind, the state of events and the fulfillment of an Event Condition Set can be understood by the following rules.

- a. All Conditions are initially OFF when the system is loaded. Subsequently, they receive values according to the following:
  1. When any of the Events associated with a Condition is signalled ON either by a Task or by the Executive, the Condition is set ON.
  2. When any of the Events associated with a Condition is signalled OFF, the Condition is set OFF.
  3. Unlatched events have a life span with respect to a given Task. When a Task is SCHEDULED, all of its unlatched events are guaranteed to be unfulfilled. Upon the Task becoming ACTIVE, all unlatched events are again made to be unfulfilled.
  4. While it is SCHEDULED or while it is ACTIVE, the unlatched event conditions are being accumulated in order to fulfill the Event Condition Set. They are only reset as indicated above.
- b. When all the Conditions of a Task which is INVOKED but INACTIVE have their desired values, the Task becomes ACTIVE.

#### 4.4.3.6 Time

The DAIS Application Software may reference time in three ways: as Mission Time, as Relative Time, or as Cyclic Time. Mission Time is a count of elapsed Minor Cycles since system initialization. Relative Time is a count of Minor Cycles in advance of the point at which the reference to time is made. Cyclic Time is used to specify synchronous actions. It is referenced in terms of period and phase. Period is the number of Minor Cycles between successive occurrences of the action. The maximum specifiable period is a Major Frame. Phase is the offset of the first occurrence of the action within a Major Frame from the start of the Major Frame, measured in terms of Minor Cycles.



Thus, for instance, an action specified as period=16, phase=3 will occur on Minor Cycles 3, 19, 35 ... . It is necessary that  $0 \leq \text{phase} < \text{period}$ .

#### 4.4.3.7 Real-Time Interfaces

The Local Executive provides the interface to each of the Real-Time constructs. These are the:

- Real-Time Declarations
- Real-Time Built-in Functions
- Real-Time Statements
- Real-Time Directives.

While the first three interfaces provide access to the Executive routines, the Real-Time Directives change the semantics of the other Real-Time Constructs. These directives were added from the baseline design in order to provide execution improvement. Their use is discouraged. In the case of the Local Copy Override, it is in general forbidden.

The semantics of these directives are as follows:

<privileged mode task directive> ::= PRIVILEGED'MODE'TASK;  
<local copy override directive> ::= LOCAL'COPY'OVERRIDE

4.4.3.7.1 Privileged Mode Directive - The Privileged Mode Directive places Tasks in the Privileged Mode. It enables or disables certain Real-Time constructs as shown in Table 9.

Statement	Action
Local Copy	Disable
Global Copy	Enable
Read	Disable
Write	Disable
Access	Enable
Broadcast	Enable
Read Device	Disable
Write Device	Disable
Trigger	Disable

Table 9.  
Result of Privileged Mode Directive

4.4.3.7.2 Local Copy Override - The Local Copy Override Directive allows Normal Mode Tasks to have direct access to the Global Copies of Compool blocks, while still using Local Copy declarations. The use of this directive defeats the protective mechanisms supplied by the Executive when a Task references a Compool block. Its use should be in general forbidden.

#### 4.4.3.8 Master Executive Interfaces

4.4.3.8.1 Startup/Loader - The hardware interfaces for the Startup/Loader are:

- The Mass Memory, from which it reads programs to be loaded
- The Processor Control Panel (PCP) from which it reads information to determine which functions it will perform
- The BCIU. The Startup/Loader will set the BCIU registers.

AD-A085 136

INTERMETRICS INC DAYTON OH

F/8 9/3

DIGITAL AVIONICS INFORMATION SYSTEM (DAIS): MISSION SOFTWARE.(U)

FEB 80 S F STANTEN, P Y WILLIAMS

F33615-75-C-1181

AFWAL-TR-80-1003

NL

UNCLASSIFIED

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

2-2

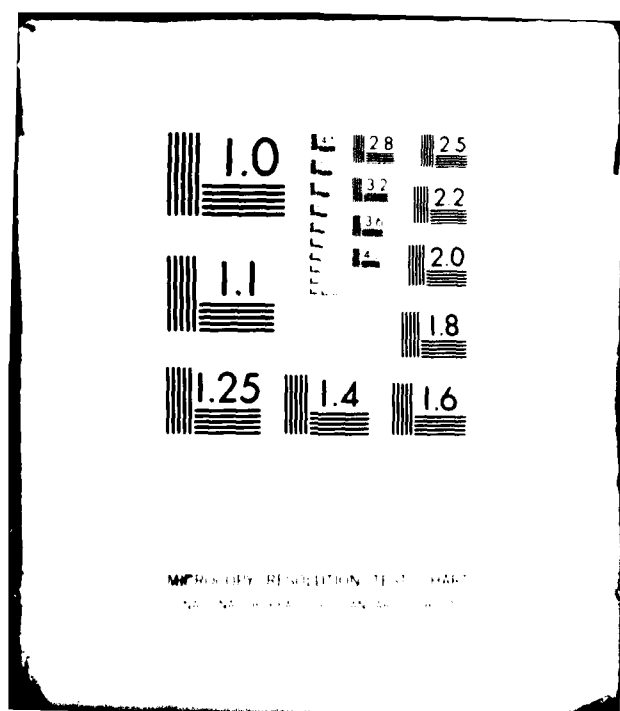
END

DATE

FILED

7 80

DTIC



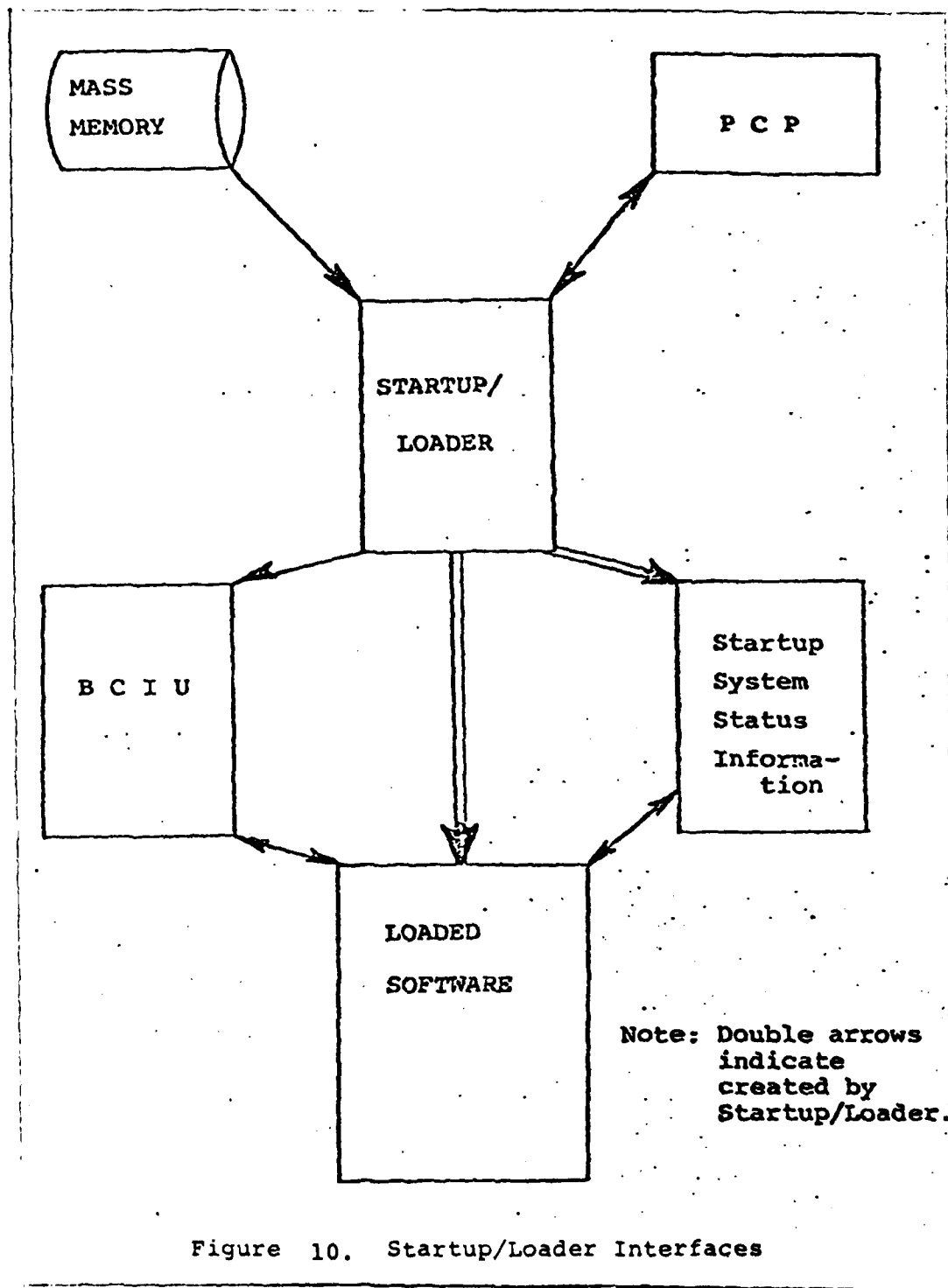
The Software interfaces through the table the Startup/Loader creates, which contains the address at which to start processing and the storage protection keys.

These relationships are shown in Figure 10.

4.4.3.8.2 Master Initialization - Master Initialization receives control from the Startup/Loader. It initializes certain Executive Tables in its processor. These Tables are set to schedule both the Master Sequencer and a task to perform CPU hardware tests. These relationships are shown in Figure 11.

4.4.3.8.3 Bus Control Interfaces - Bus Control is the part of the Master Executive which controls sending messages over the DAIS Bus. Figure 12 shows the interfaces of the various parts of the Bus control (shown in the double line boxes) with the rest of the system. Each of the elements is discussed below.

- a. Bus Controller - This function sets the BCIU registers for sending the Bus messages.
- b. Critically Timed Message Processing - Critically Timed Message Processing receives messages from either the Local Executive or Error Processing. It also processes the Master Executive Minor Cycle event. It sets Timer A to interrupt at the proper mission time to send this message. When Timer A interrupts, Critically Time Message Processing checks the type of message at the top of its queue. If this message is a normal type, that is, a critically timed message scheduled by the Local Executive, the Bus Controller is called to send the Message. If the Message at the top of the queue is the Master Executive Minor Cycle Event, then the Minor Cycle Check function will be called.



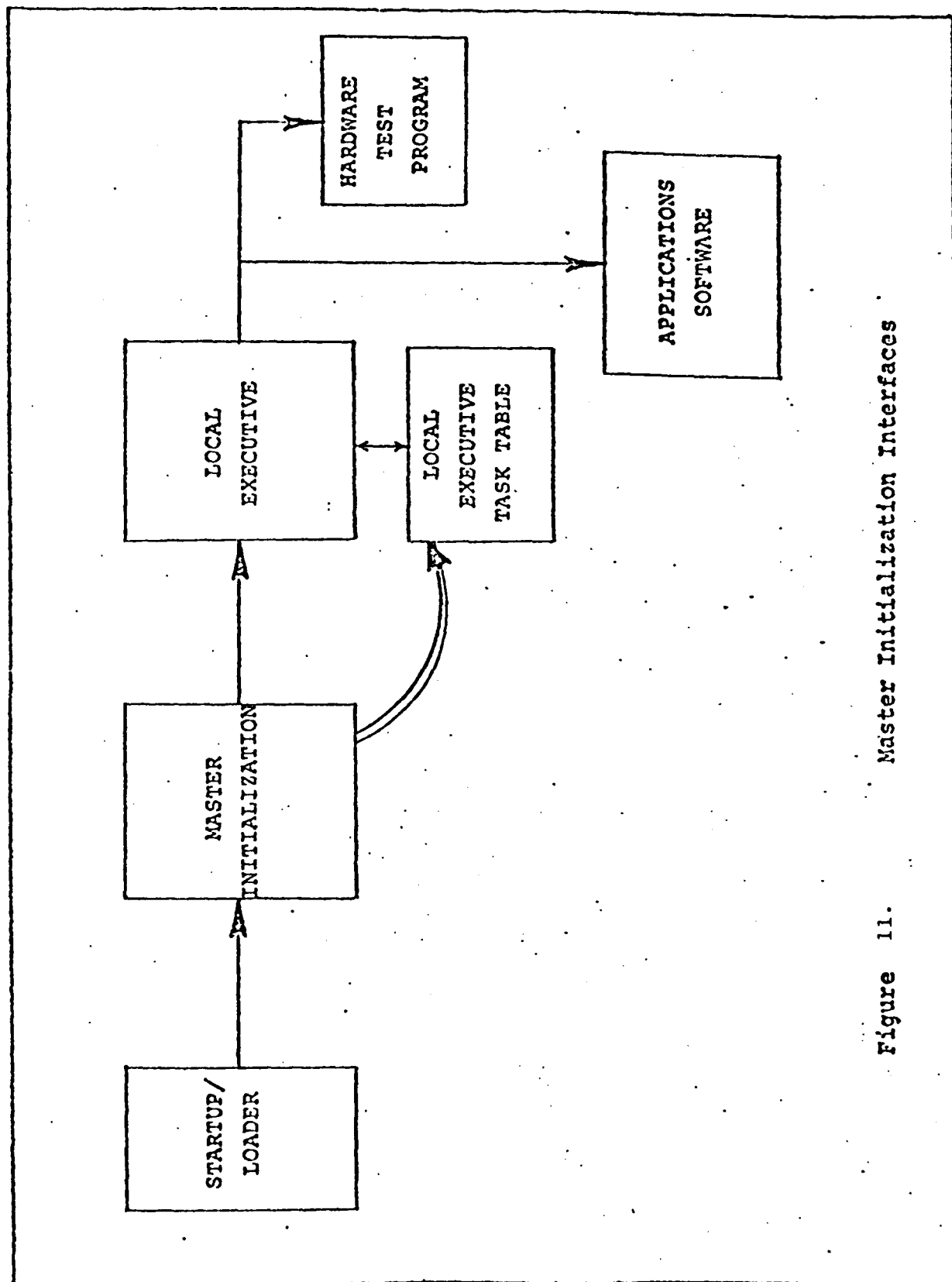


Figure 11. Master Initialization Interfaces

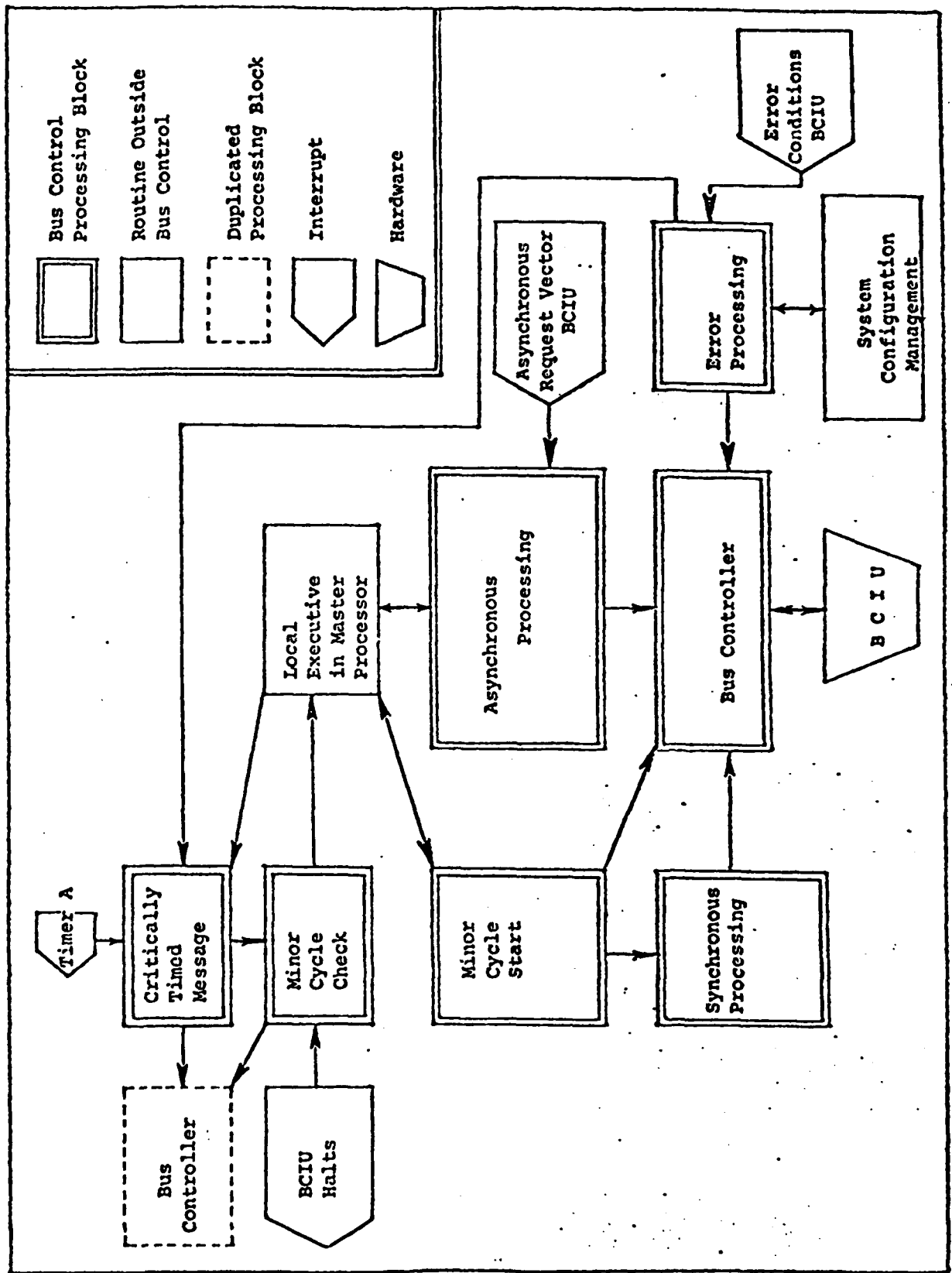


Figure 12. Bus Control



- c. Minor Cycle Check - Minor Cycle Check is invoked by a BCIU Halt, by calls from the System Startup (as part of the starting sequence), or by Critically Timed Message Processing.

During system initialization, Master Function Mode Command generation does not start until the Master Sequencer has performed its initialization. During normal operation, Master Function Mode Commands must wait for a BCIU Halt (the end of the previous cycle's synchronous bus list processing), and an indication from Critically Timed Message Processing (indicating it is time for a new Minor Cycle). If a Minor Cycle should start, the Bus Controller is called to send Master Function Mode Commands to all Remote Processors and then transfer control to the Master Processor's Local Executive to start a Minor Cycle.

- d. Minor Cycle Start - After the Local Executive in the Master Processor has done its Minor Cycle Processing, it calls Minor Cycle Start. Minor Cycle Start then waits until all other processors have received the Master Function Mode Command then starts Synchronous Processing. It returns control to the Local Executive.
- e. Synchronous Processing - the Synchronous Instruction List (SIL) is obtained for this Minor Cycle. Then control is passed to the Bus Controller to be started. When Synchronous Processing is complete, the SIL Done Event is set ON.
- f. Asynchronous Processing - Asynchronous Processing receives Asynchronous Request Vectors either from the Local Executive in the Master Processor or from BCIU interrupts. The message to be sent is found and sent by the Bus Controller. Asynchronous Processing may also invoke the Local Executive for an Asynchronous Message received or transmitted so that the Local Executive may perform the necessary processing.

- g. Error Processing - Error Processing is always invoked by a BCIU generated Interrupt. Examination of the BCIU registers will indicate the cause of the error and indicate the action to be performed. A Bus error may indicate that a System Component has failed. If so, System Configuration Management will be called to handle the problem.
- h. Bus Control in Remote Processors - Figure 13 shows the relation of Bus control to the other processors. The BCIU acts as the extension of the Master Executive. The Local Executive sets the Status Code Register when it wishes to send an Asynchronous Message. When the message has been sent, the Local Executive will be interrupted. A similar interrupt occurs when an Asynchronous Message is received. A different interrupt occurs to denote the Master Function Mode Command, which signifies the start of a new Minor Cycle.

4.4.3.8.4 Monitor/Backup Interfaces - Monitor/Backup is that software which resides in the Monitor Processor in order to determine when to assume control.

The Monitor/Backup interfaces with the Master Executive through messages received across the DAIS Bus. Figure 14 shows this inter-relationship.

The Monitor is waiting for the Master to fail. The Master Executive has to send the Monitor a message that it has not failed, otherwise, the Monitor will take control of the DAIS System. The Monitor may also be invoked by the applications software. This Backup action is initiated if the applications code detects an error and wishes to force a Monitor takeover. In addition, any permanent hardware errors detected by the System Configuration Management must also be sent to the Monitor.

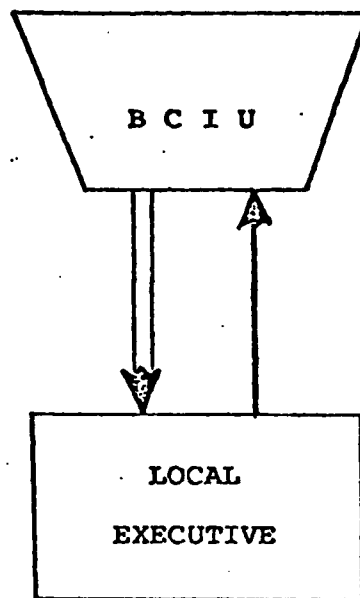


Figure 13.

Bus Control in Remote Processors

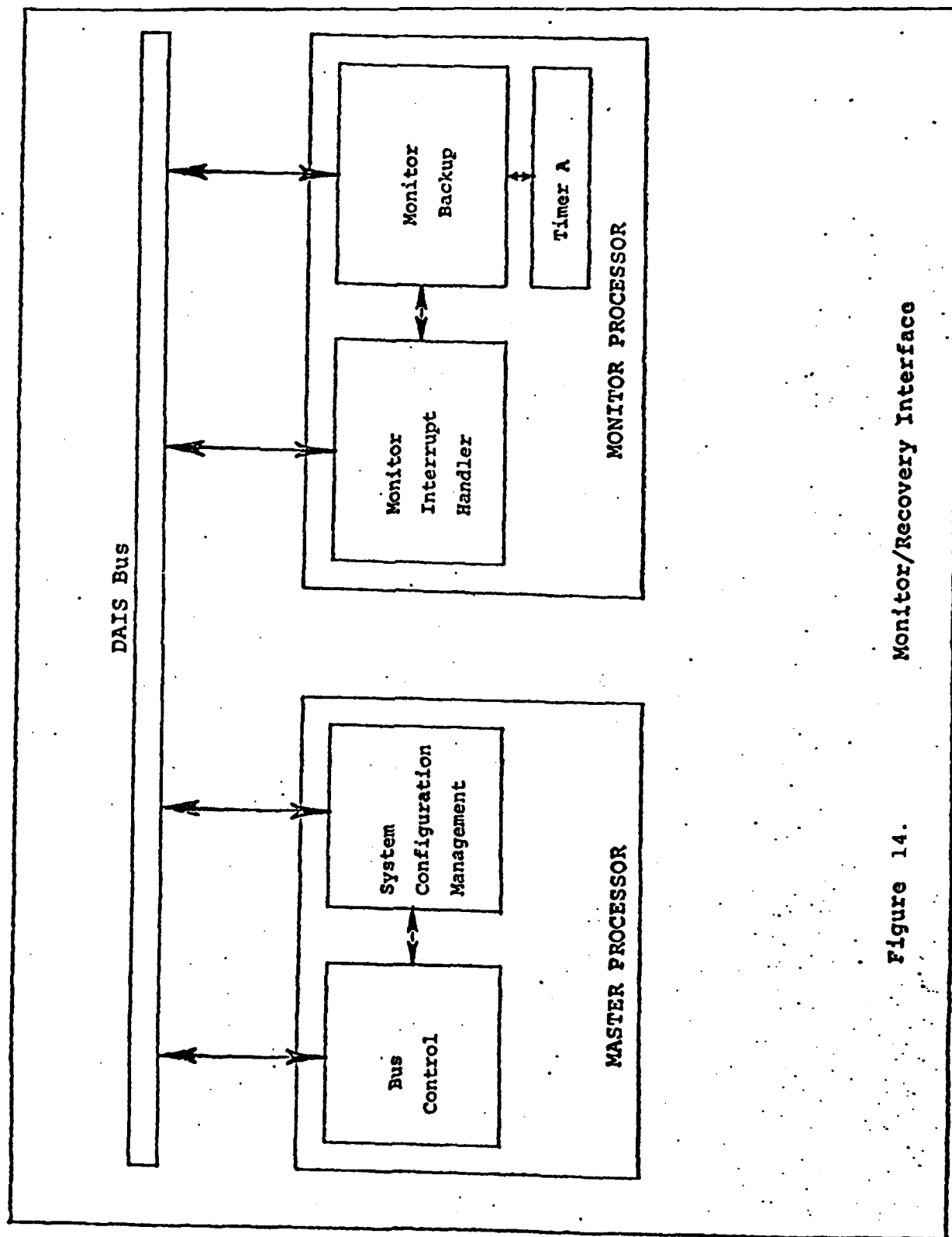


Figure 14. Monitor/Recovery Interface

#### 4.4.3.8.5 System Configuration Management Interfaces - System Configuration Management interfaces with:

- a. The Bus Control to detect permanent device failures and take corrective action.
- b. The Monitor to inform it of permanent device errors.
- c. The Local Executive, which generates error indications for Local Executive detected conditions.
- d. The Master Sequencer, in order to initialize or reinitialize Applications Software.
- e. The Applications Software, when a permanent device failure within an RT is detected by Bus Control, System Configuration Management informs the Applications Software of this fact.

Figure 15 shows the System Configuration Management interface relationships. If the Applications Software no longer wishes to receive messages from certain RT subaddresses, it informs the Subsystem Status Monitor. Usually, this means the Applications Software has detected a failure within the remote terminal subaddress and no longer wants that data. Applications Software may also detect an unrecoverable condition and inform System Configuration Management of this condition. In this case, System Configuration Management will either restart the Master Sequencer or invoke Monitor/Backup.

#### 4.4.3.8.6 Reconfiguration Interfaces - Reconfiguration receives its control from two separate inputs:

- The System Configuration Management Indication that at least one active processor has failed.
- Data from the PCP.

Reconfiguration processing controls the loading and initialization of the new configuration, and is initiated upon pilot request.

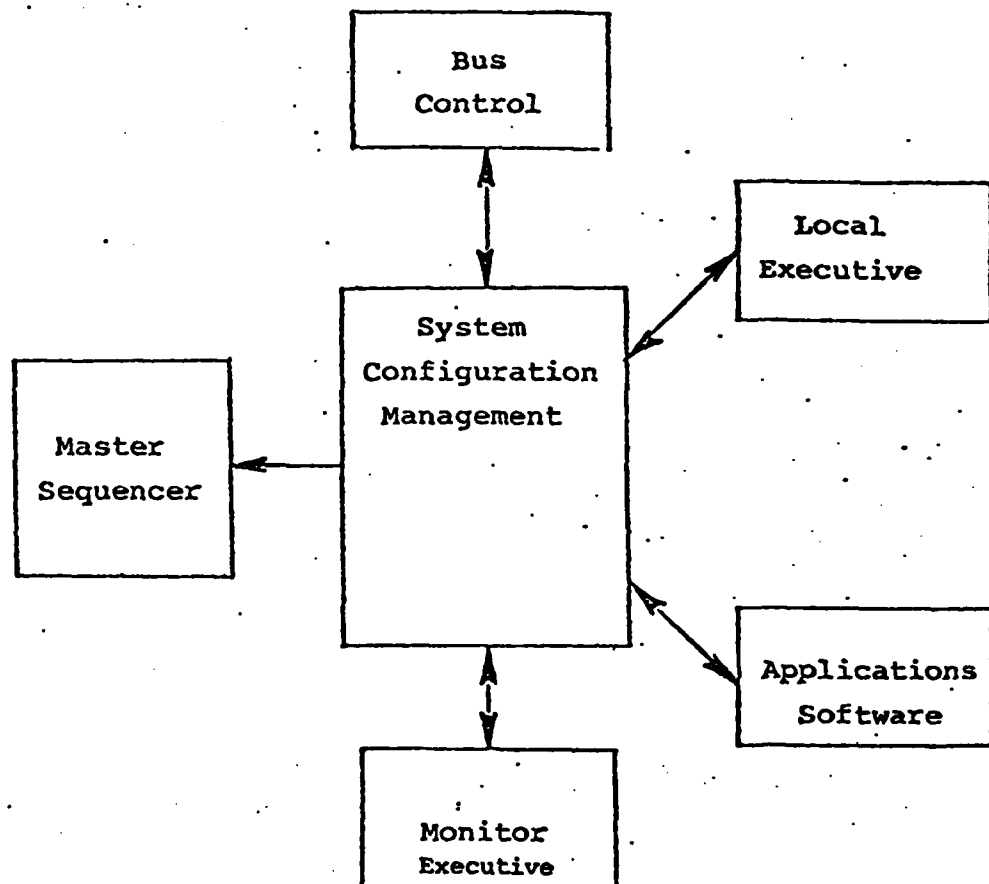


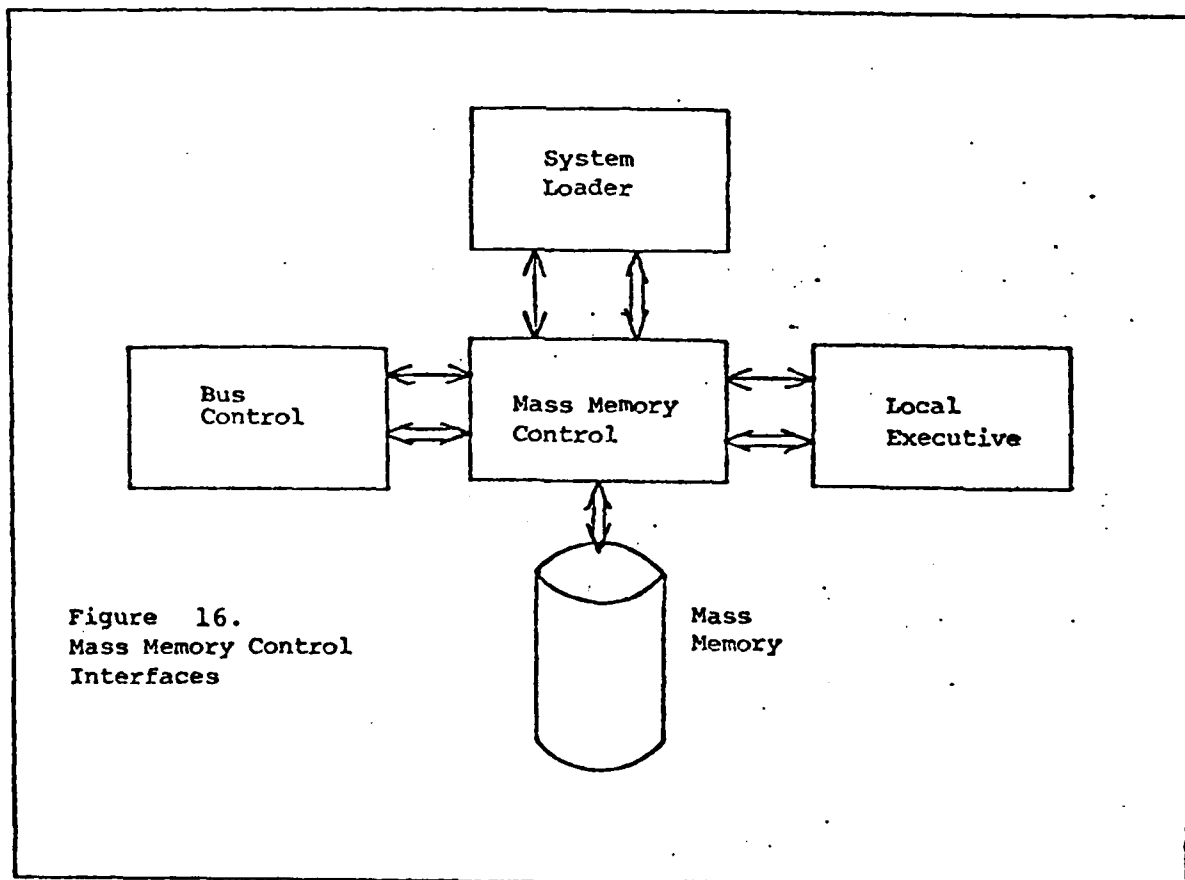
Figure 15.

System Configuration Management  
Interfaces

4.4.3.8.7 Mass Memory Control Interfaces - Mass Memory Control Interfaces with:

- The Bus Control to manage the bus protocol of interrupts.
- The Local Executive to receive indications of requested data transfers.
- The System Loader to load all active processors within a configuration from mass memory.

Figure 16 shows the interface relationships. No request of Mass Memory Control may exceed the thirty-one word limit which is imposed by the MIL-STD 1553A bus protocol and DAIS asynchronous protocol.



#### 4.4.4 Hardware Interfaces

The Executive Software must interface with particular pieces of hardware. In the DAIS system, these consist of:

- DAIS Multiplex Data Bus
- Remote Terminals
- Bus Control Interface Unit
- DAIS Processor: AN/AYK-15
- Mass Memory
- Processor Control Panel
- Advisory Caution Lights

#### 4.4.5 Interface to PALEFAC

The DAIS Executive must interface with two elements of non-Real-Time software, PALEFAC and the Language Translators.

PALEFAC provides the Executive Tables which are the primary data base of the DAIS Executive. These tables include:

- The Tasking Tables, which describe the states and inter-relations of Tasks and Events.
- The Compool Area, which contains the Global Copies of Compool Blocks.
- The Data Descriptor Blocks, which describe the Compool blocks.
- The I/O Tables, which control Synchronous and Asynchronous Data Bus traffic.

The PALEFAC-produced Executive Tables are described in detail in the PALEFAC Pre-Processor/PALEFAC-Mission Software Interface Control Document. The tables generated are specified in the detail design of the Executive Software (Local Executive and Master Executive) requirements.



#### 4.5 Applications Software

The DAIS Applications Software requirements for the OFP are derived from existing aircraft implementations but contain some significant differences reflecting the innovative controls and displays that DAIS is designed to implement.

The software structuring of the DAIS Operational Flight Program differs from that of most existing fleet aircraft for a number of reasons. First, the top-down structuring of the DAIS OFP supports a structure for future functional expansion with minimal cost impact. Secondly, the structure of the DAIS OFP allows for easy transferral of the software to any machine that supports J73/I. Thirdly, the DAIS OFP had been coded to exploit the facilities and capabilities of the DAIS Executive system as presented by the DAIS Real-Time Interface.

An overview of the major functions of the DAIS Applications Software will be presented. The interfaces to flight hardware and to the DAIS Executive software are delineated, and the interaction between the elements of the applications software is described.

##### 4.5.1 Equipment Interface

The DAIS Applications Software is designed to interface to sensor, pilot control, display and mass memory unit hardware. Table 10 delineates the Mission & Configuration and shows the various equipments that required interfacing

##### 4.5.2 Software Interface

The DAIS Applications Software executes in an environment maintained by the DAIS Executive software. The Executive provides real-time task control, data base management, interprocessor communication, remote terminal communication and mass memory services. The Mission Software is designed to minimize the complexity of these operations to the Applications programmer. A description of the applications/executive interface from a programming standpoint is contained

TABLE 10. MISSION & CONFIGURATION

Weather:	Night - Clear (Visual Flight Rules [VFR])																
Target:	Fixed Ground Target																
Weapons:	MK-82 LDGP Bombs																
Threats:	None																
Simulated Sensors:	INS (SKN2416) Laser Ranger Air Data Sensors Radar Altimeter (APN-141) ILS (ARN-58A) TACAN (ARN-118)																
Core Element (Hardware):	DAIS Processors (2): Master & Remote #1 BCIUs (2) RTs (2) Controls and Displays <table border="0"> <tr> <td>RT (1)</td> <td>C&amp;D Mass Memory</td> </tr> <tr> <td>VSD</td> <td>SCU</td> </tr> <tr> <td>HSD</td> <td>HUD</td> </tr> <tr> <td>MPD-1</td> <td>PCP</td> </tr> <tr> <td>MPD-2</td> <td>MPDG (1)</td> </tr> <tr> <td>IMFK</td> <td>DSMU</td> </tr> <tr> <td>DEK</td> <td>AP</td> </tr> <tr> <td>MMP</td> <td></td> </tr> </table>	RT (1)	C&D Mass Memory	VSD	SCU	HSD	HUD	MPD-1	PCP	MPD-2	MPDG (1)	IMFK	DSMU	DEK	AP	MMP	
RT (1)	C&D Mass Memory																
VSD	SCU																
HSD	HUD																
MPD-1	PCP																
MPD-2	MPDG (1)																
IMFK	DSMU																
DEK	AP																
MMP																	
Support Facility:	Integrated Test Bed																
Functions:	Navigation - Inertial/Baro-Damped  Steering - Command NAV <table border="0"> <tr> <td>TACAN</td> </tr> <tr> <td>ILS</td> </tr> </table> Navigation Update - Flyover <table border="0"> <tr> <td>HUD/Laser</td> </tr> <tr> <td>Ranger</td> </tr> <tr> <td>FLIR/Laser</td> </tr> <tr> <td>Ranger</td> </tr> </table>	TACAN	ILS	HUD/Laser	Ranger	FLIR/Laser	Ranger										
TACAN																	
ILS																	
HUD/Laser																	
Ranger																	
FLIR/Laser																	
Ranger																	

TABLE 10. MISSION & CONFIGURATION (con't.)

Acquisition/Cueing - Pilot/HUD  
Pilot/FLIR

Target (or OAP) Fix -  
HUD/Laser Ranger  
FLIR/Laser Ranger

Weapon Delivery - CCIP/Auto  
CCIP/Manual

Stores Management

Communications - UHF

Checklist

in the DAIS Software Standards and was presented in Section 4.3. The fact that a multiple-processor operation exists is entirely transparent to the programmer on the software module level. The partitioning of the software into multiple processors is accomplished by the use of the PALEFAC facility.

#### 4.5.3 Applications Software Architecture

The DAIS Applications Software is structured as a fixed invocation tree. The software elements in this tree are of four types:

- System Control Modules - These tasks are responsible for the control and initialization of the rest of the applications software tasks.
- Specialist Functions (SPECs) - These modules accomplish specific computational tasks associated with one of the functions of navigation, guidance, weapon delivery or stores management.
- Display Functions (DISPs) - These processors control the operation of the cockpit displays.
- Equipment Processors (EQUIPs) - These tasks interface with the DAIS sensors and controls.

##### 4.5.3.1 System Control Modules

System Control Modules control and initialize the applications tasks as follows:

- The top task in the DAIS Applications invocation tree is the Master Sequencer. This task schedules the Configurator, the IMFK/MFK and MMP Request Processors, and the Master Mode Panel (MMP) lights.
- Configurator - Controls the operations of the applications programs (SPECs, DISPs, EQUIPs).

- Request Processor - Interprets pilot inputs from the panels, specifically the Master Mode Processor (MMP), Integrated MultiFunction Keyboard (IMFK), and MultiFunction Keyboard (MFK).
- Subsystem Status Monitor - Keeps track of the status of the avionics subsystems (i.e., equipment).
- IMFK Handler - Services inputs from menu keys on the IMFK panel.
- MFK Handler - Services inputs from menu keys on the MFK panel.

#### 4.5.3.2 Specialist Functions (SPECs)

A SPEC is a task that carries out supporting computational functions associated with a master mode. The different SPECs and their definitions are as follows:

- Navigation Computation SPEC - responsible for keeping track of the aircraft navigation state (latitude, longitude, wander angle, altitude, attitude, wind, and velocities), utilizing information from the Inertial Navigation System (INS), the Air Data Computational SPEC and the various fixes.
- Air Data Computational SPEC - generates barometric altitude, True Air Speed, calibrated Mach Number, and static temperature for other computation SPECs and/or displays utilizing information from the air data sensors.
- Guidance Computational SPEC - provides steering cue data for the displays; in particular, it positions the flight director to facilitate steering to waypoints, to a heading, or altitude and during ILS landing.

- Stores Management SPEC - includes stores setup, stores inventory maintenance, and weapon release processing.
- Weapon Delivery SPEC - performs all processing required for the execution of the Weapon Delivery mode selected by the pilot, including algorithmic execution, control of sensors and management of displays.

#### 4.5.3.3 Equipment Processes (EQUIPs)

EQUIPs are tasks that interface with the DAIS sensors and controls. Each piece of equipment is communicated with by one or more input EQUIPs (from the equipment to the software) and/or one or more output EQUIPs (from the software to the equipment).

EQUIPs were introduced into the DAIS software structure in order to separate the details of communication with equipment from the algorithmic and logical functions performed by the software. That is, if the details (formats) of an equipment change, but its function remains the same, then only the EQUIP software need be modified.

Input EQUIPs receive messages from the external equipment via a remote terminal. These messages are then converted to internal form. Output EQUIPs read the output from other processing modules, format the output for the equipment, and output the data to the external equipment via a remote terminal. The equipments for which EQUIPs have been isolated include:

- Inertial Navigation System (INS)
- Laser Ranger
- Instrument Landing System (ILS)
- TACAN
- UHF
- Pave Penny
- VATS/Pave Tack

- Radar Altimeter
- Air Data
- Engine, Fuel, Flaps and Speed Brake Status Systems
- Station Logic Unit (SLU) and Weapon Stations
- Armament Panel
- Data Entry Keyboard
- Sensor Control Unit

#### 4.5.3.4 Display Processes (DISPs)

DISPs are tasks that control the cockpit displays. More complex displays, Head Up Display (HUD), Vertical Situation Display (VSD), Horizontal Situation Display (HSD), MultiPurpose Display (MPD), are generated by the Modular Programmable Display Generator (MPDG). A DISP receives input from various mission tasks, formats messages to control the display and outputs them to the MPDG or the display device through the remote terminals.

The equipment for which DISPs exist are:

- Integrated MultiFunction Keyboard (IMFK)
- Master Mode Panel (MMP)
- MultiFunction Keyboard (MFK)
- Sensor Control Unit (SCU) lights
- MultiProgrammable Display Generator (MPDG)
- Moving Map Device (MMD)

#### 4.5.4 Software Interactions

This section presents examples of the DAIS Software's interactions between the elements introduced above in Section 4.5.3.

##### 4.5.4.1 IMFK/MFK - Pilot Interface

Most pilot-initiated functions, except those on the highest level, are activated through the IMFK. Each OPS and each Brute Force SPEC have one or more associated IMFK pages,

each of which has up to ten items. By pressing the key associated with one of the items on an IMFK page, the pilot may activate a software function.

There are three types of IMFK pages: Checklist pages, Tailored pages, and Brute Force pages. MFK pages are Brute Force pages.

- a. Checklist Pages - This type allows a pilot to complete a checklist via the IMFK. Each checklist consists of one or more checklist page(s). There are two types of checklist pages: those on which every key must be pressed, and those which allow advancement to the next page by pressing key 10. Items on these pages are of four types:
  1. Check items - The pilot checks the status of some piece of equipment and presses the key indicating he checked the item.
  2. Data entry items - Each of these items allows the pilot to change the values of one or more mission data variables by entering new values through the Data Entry Keyboard (DEK).
  3. Action Items - These items start or stop various functions or equipment.
  4. Advance page items - These items cause the checklist to advance to the next page.

Checklists occur during the preflight, cruise, approach and landing, precision approach, and de-arming phases and may also be called up through the Checklist Brute Force SPEC.

- b. Tailored Pages - There is one tailored pages associated with each master mode except Preflight. Each of these pages allows the pilot to choose several functions relevant to the associated master mode. A tailored page remains on the IMFK throughout most of the master mode, and its items can be chosen at any time. Items on these pages are either data items or action items.



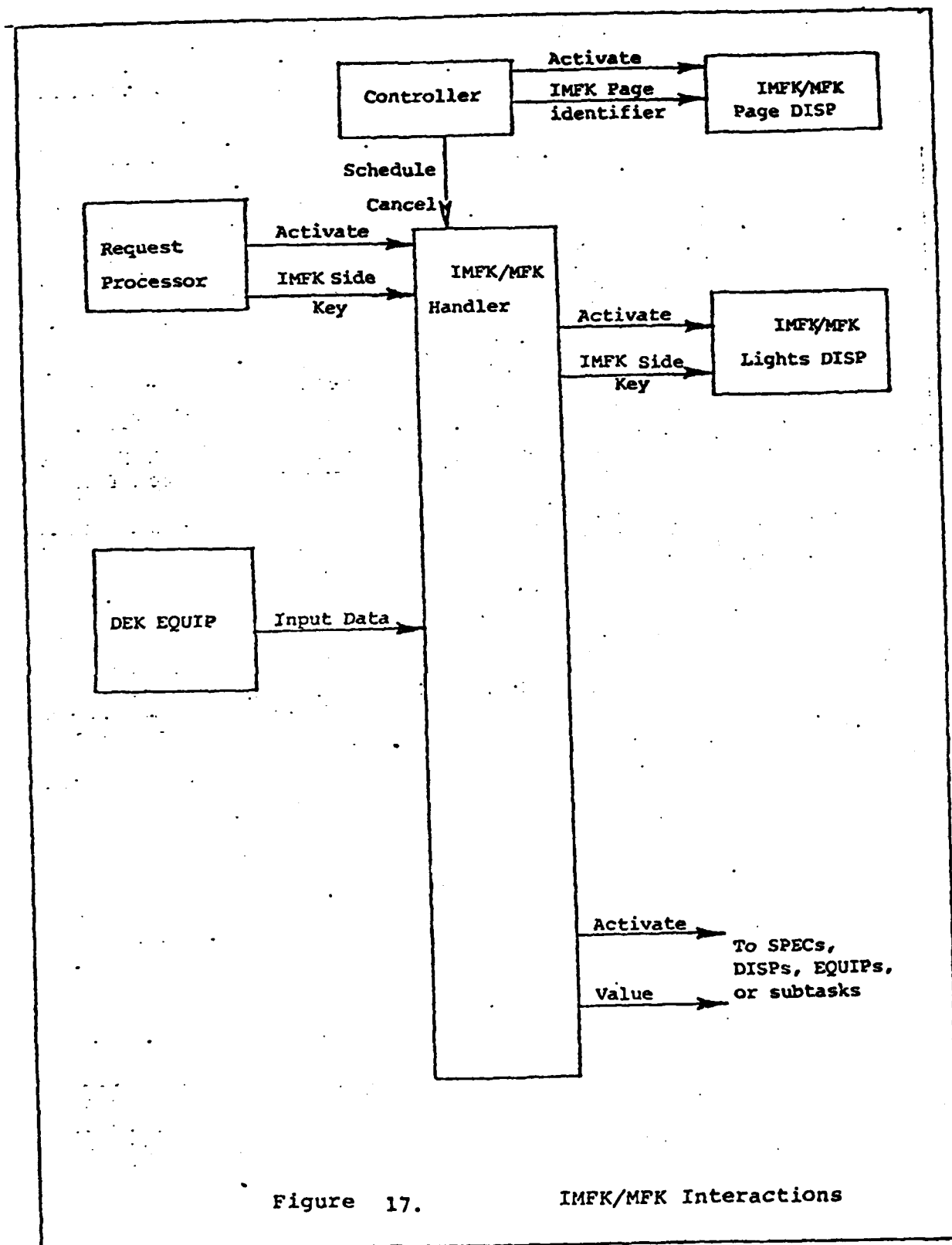
- c. Brute Force Pages - Several Brute Force pages are associated with each Brute Force SPEC. Each group of pages is part of a tree structure. The highest level page of a group is displayed. An item chosen on this page will either cause a function to be carried out, or will cause a next (2nd) level page to be displayed. An item chosen on a 2nd level page will either cause a function to be carried out or a 3rd level page to be displayed. All items on 3rd level pages will cause functions to be carried out. Thus, as many as four keys may have to be pressed to invoke a function.

Figure 17 shows the interactions of an IMFK/MFK handler task with the Request Processor, the DEK EQUIP, and the IMFK DISPs and the MFK DISPs. The following sequence of interactions occurs.

The Controller (any OPS or Brute Force SPEC) activates the IMFK/MFK Page DISPs to display a new IMFK/MFK page.

When an IMFK/MFK menu key is pressed by the pilot, the Request Processor activates the IMFK/MFK Handler and passes it the number of the side key. Depending on the type of item, one of the following sequences is carried out by the IMFK/MFK Handler.

- Check Item - The IMFK/MFK Handler places a 'checkmark' next to the item to mark the item as completed.
- Data Entry Item -
  - The IMFK or MFK Lights DISP is activated to backlight the appropriate key.
  - The DEK is activated.
  - Input data from the DEK EQUIP is awaited. If the pilot decides not to enter any data, he must at least press the DEK ENTER key.
  - The pilot's input data is sent from the DEK EQUIP to the IMFK/MFK Handler.



- The DEK is deactivated.
  - If input data were received, various SPECS DISPs, and EQUIPs that use the data may be activated and sent the value, and the the new value is displayed as part of the IMFK item.
  - The IMFK or MFK Lights DISP is activated to turn off the side key light.
- Action Item - One or more SPECS, DISPs, EQUIPs, or subtasks are activated to carry out the desired Function.
  - Advance Page Item - When a checklist page is complete or a brute force item is pressed, the IMFK or MFK Page DISP is activated to display the new page.

#### 4.5.4.2 Navigation Interfaces

The Navigation function receives inputs from the navigation sensor EQUIPs and outputs a set of data (the Navigation State) to the DISPs, Weapon Delivery and Guidance functions.

Navigation data is typically expressed with respect to two coordinate frames. The local-level frame is defined as having its origin at the position of the aircraft, with the axes located in the plane tangent to the earth's surface and orthogonal to each other, with the z-axis directed away from the earth, orthogonal to the tangent plane. The Body (or Aircraft) frame is defined as having the y-axis point out the nose of the aircraft, x-axis out the right wing, and the z-axis orthogonal to the x and y forming a right-handed cartesian coordinate system. While these coordinate frame definitions are not standardized across the available sensors and software systems currently in use, it has been decided to make the coordinate definitions above standard to DAIS

and to convert all sensor input to these standard frames of reference, and to design all subsystems interfacing with the Navigation SPEC to these coordinate frame definitions.

#### 4.5.4.3 Normal Attack Interactions

In order to accomplish the Normal Attack sequence, many elements of software must interact in a controlled manner. The purpose of this section is to indicate interactions, rather than to describe the algorithmic processing of the Normal Attack Controller. The elements conceptually interact as follows:

- Upon depression of the CCIP/AUTO MMP key, the Configurator will schedule and activate the Weapon Delivery OPS. This is a generic OPS under which one of many possible bombing modes may be controlled. In this example, the Weapon Delivery OPS activates the Normal Attack Controller.
- The Normal Attack Controller (NAC) is responsible for controlling the sequencing and algorithmic calculations involved in this mode of Weapon Delivery.
- NAC also sends data to the MPDG DISP so that the required graphic and numeric data can be displayed.
- The SCU mode (i.e., LSR) is sent to the SCU Lights DISP and the SCU Bullpup Controller EQUIP is activated. Information from this EQUIP is used to drive the Aiming Reticle as well as the LSR.
- The Designate Button EQUIP causes NAC to switch from target acquisition to weapon delivery. This implies that the FLR RANGE EQUIP must be used to read the target range so that solution cues can be generated.

- Alternate targets, or a refined target position, may be obtained by using the bullpup controller to move the AR and FLR. Depression of the Designate button causes a reinitialization of the Weapon Delivery processing.
- The SLU EQUIP is activated only when the calculated time-to-go is less than a preset value, the master arm is set, and the Armament Release Button (ARB) is depressed. The event EARB or a Boolean tag is set by the Armament Release Button EQUIP.

#### 4.5.4.4 Interactions of Input EQUIP Functions

This section describes the generic interactions between an input EQUIP function (EQUIP), the I/O Compool data (IN), the task using the sensor data (TASK), the Subsystem Status Monitor (SSSM), the Configurator, and the display functions associated with the equipment status (DISP).

The Configurator schedules TASK, EQUIP and the DISP according to the system configuration requirements. The basic function of the EQUIP is to read sensor data, reformat it and place it in the compool. TASK and DISP use reformatted sensor data as required.

An auxiliary function of EQUIP is to perform equipment dependent limit checks on the sensor input data. If a data element is out-of-range, an error message is formulated and sent to the Subsystem Status Monitor (SSSM). It is the function of SSSM to count error messages, gather statistics, and determine when an equipment is to be declared failed. The input EQUIP signals each potential error. It does not remember error statistics. All error history is maintained by the SSSM.

## SECTION V. DEVELOPING A MISSION

The process of implementing a specific avionics mission using the DAIS methodology and tools requires both the development of the software and the simulation and testing of the software. Both of these aspects will be discussed below and will be shown to be an outgrowth of the actual DAIS Mission Software design and standards.

Having discussed the details of the DAIS Executive interface to the Applications Software in Section 4.0, a careful consideration of how one would like to develop avionics software should be taken. An OFP could be broken down into numerous functional structures. A reasonable breakdown could consist of two major parts; one part the Executive Software and the other part the Applications Software. In this functional breakdown the Executive Software can be thought of as that software resident within the processors that is mission invariant. The Applications Software is that which could change from mission to mission, i.e., a close air support mission versus an air superiority mission, or from aircraft to aircraft, i.e., dependent on the sensors and subsystems.

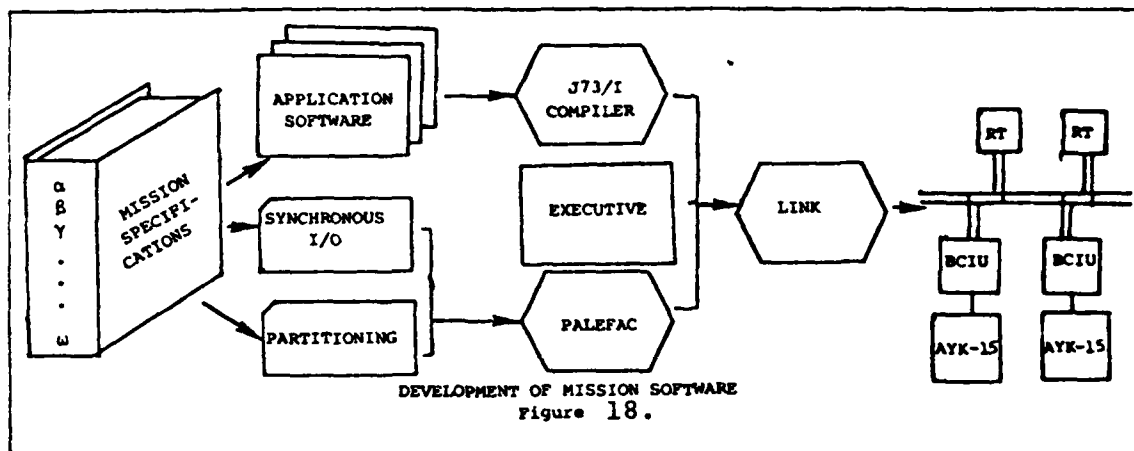
If it is accepted that the Executive Software is operationally mission independent, then to develop a new OFP should consist of writing Applications Software according to the mission specifications, describing the equipment suite I/O interface, and describing the computer network. A pictorial representation of this method is shown in Figure 18. The specification of synchronous I/O in the figure is a reasonable and convenient way to provide global information. Since DAIS is based upon the concept of a federated computer systems, partitioning of global information is also required.

Considered from this point of view, there are various characteristics that would be desirable for Mission Software to have in addition to processor invariance and an automatic solution to potential data conflicts. A list of such goals for Applications Software would include:

- Invariance with respect to processor and I/O control
- Invariance with respect to executive implementation
- Invariance with respect to partitioning across processors
- Invariance with respect to network
- Automatic Multiprocess Synchronization for:
  - Data Conflicts
  - I/O
  - Interprocessor Communications.

If it were possible to achieve these goals, then it would also be possible to initially develop and debug Applications Software on a large host computer system with confidence. In addition, the Applications Programmer would be able to develop his programs as if he were writing for a single virtual machine. While choosing a set of executive primitives the way DAIS has done does not ensure that these goals can be achieved, the methodology at least facilitates and indeed encourages such goals.

Figure 18 is accurate for the DAIS system, and the implementation of the DAIS executive has been such that the goals for Applications Software have been accomplished. In DAIS, Applications Software can be developed independent of the target processor, I/O control characteristics, the details of the executive, and indeed of the final partitioning across processors of the federated system. In addition, the implementation chosen for the Executive is such that Data Conflicts, I/O, and Interprocessor Communications are automatically handled without intervention of the Applications Programmer.



### 5.1 Mission Development

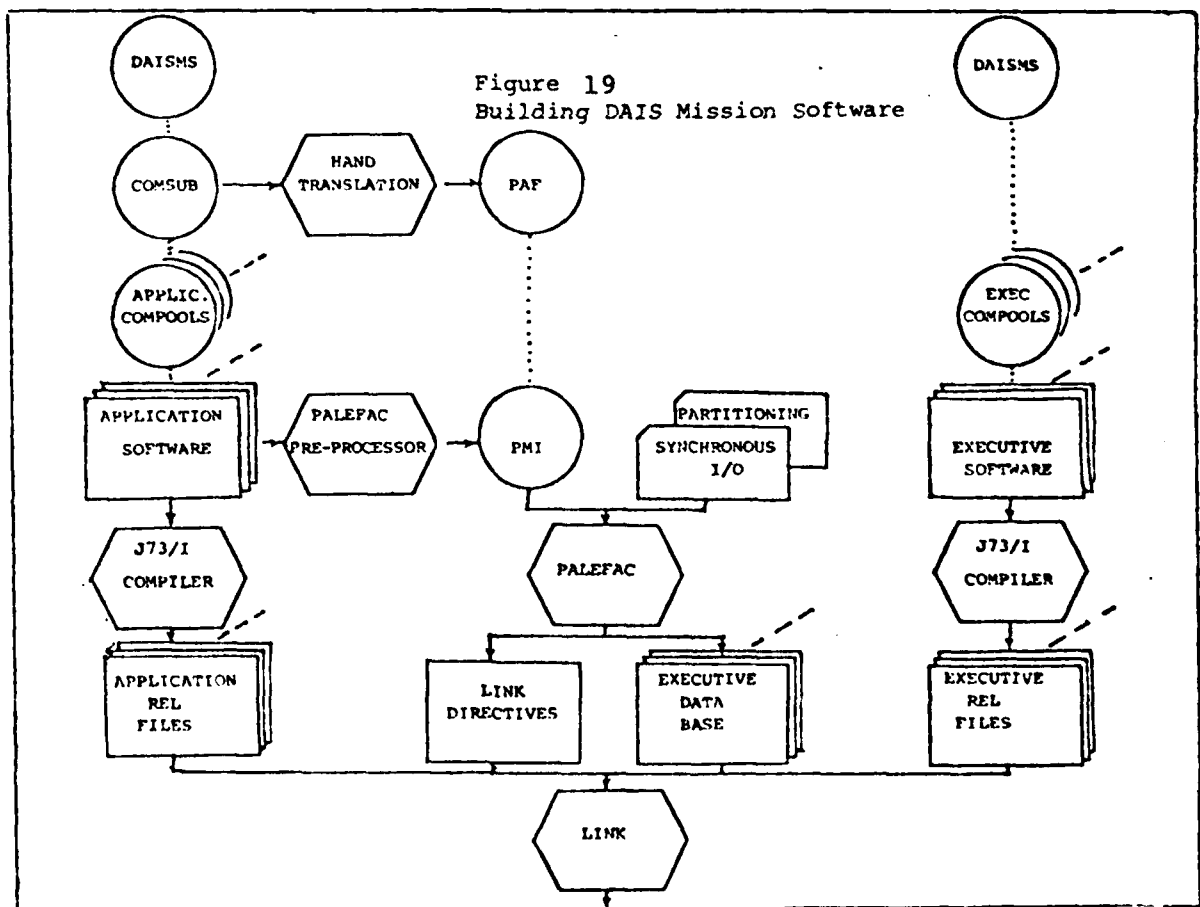
In the above discussion on how to develop a specific avionic mission, goals for the resultant Applications Software were given. Within the context of the current DAIS Executive implementations Figure 19 presents the steps and the method for generating an OFP for use with the DAIS Executive system.

The actual development of Application Software can now be seen in more detail. One Compool, named DAISMS contains the effective software standards. It includes the DEFINE interfaces to the various executive primitives and contains the necessary global information required to make the appropriate definitions and the interfaces to allow the user defined Comsubs to properly function. The Application Compools contain the user defined Compool data blocks for a given mission.

The development of the executive software is similar. These routines must also be compiled with the DAISMS Compool and their own appropriate Compools. However, once the executives are developed, they are not continually recompiled but rather the appropriate relative files are waiting to be linked.



The complexity of the required PALEFAC processing is now seen. Not only must the synchronous I/O and Partitioning information be provided, but information about both Comsubs and Tasks are required. The question regarding Task information is complex. Minimally, it is necessary to obtain the SCHEDULE statement information in order to generate the Executive Task Tables. It is also necessary to know about the LOCAL COPY declarations in order to both allocate data space, and in order to generate appropriate interprocessor data transfers for Compool blocks (if the partitioning information so indicates). Similarly, knowledge of EVENT, CANCEL, and TERMINATE is required. In order to accomplish this analysis, there is a tool called the PALEFAC PRE-PROCESSOR that reads an Application Software routine and gleans the appropriate Executive Primitives. The information is stored in a compact manner on one file.



The detailed sequence of steps required for building an avionics mission is as follows:

1. PAF Construction - Once the Comsubs have been implemented and the Compool blocks have been layed out, the PALEFAC Auxilliary File (PAF) is constructed. The PAF declares the size of all the Compool blocks, and the local storage area needed by each Comsub. The detailed syntax of this file is given in the PALEFAC User's Guide.
2. Running the PALEFAC Pre-processor - The PALEFAC Pre-processor should be run when all applications modules have been coded and successfully compiled for the target machine. The pre-processor will produce the PALEFAC Module Input (PMI) file, which is an additional input to PALEFAC. The content of the file is a condensed version of all applications tasks.
3. Compiling Applications Source Code - All applications code should be translated using the JOVIAL J73/I compiler which is resident on the DECsystem-10 at AFAL. The mechanism used for compilation is described in the JOVIAL J73/I programmers Reference Manual.
4. Running PALEFAC - PALEFAC examines the PMI file produced by the PALEFAC Pre-processor, and the contents of the PAF and the PALEFAC Global Input file (PGI). The PGI file contains information which is global to the entire configuration, such as partitioning information and I/O control information. The output of PALEFAC is the PALEFAC Mission Database (PMD) and the PALEFAC Partitioning Information (PPI). The PMD files contain all the information required by the executive for this configuration. The PPI files are link directives for this configuration.
5. The Executive - From the applications point of view the executive is already prepared for this configuration. The DAIS Systems Engineer need only include the relocatable files for the executive in the PALEFAC partitioning section of the PGI to have them configured in the load.

6. Linking - The linker for the desired target machine is invoked, provided with the PPI files as input. Exact details for the operation should be taken from the Link-10 Programmer's Reference Manual, or the LINKS Part II Specification.

## 5.2 Simulations and Testing

The development of the DAIS Mission Software required the development of various software simulation and testing tools in addition to the STS and ITB. Initially, the STS/ITB were not available, and the restrictive access and debug capability emphasized the usefulness of a host computer for simulation and testing.

### 5.2.1 Simulation Capabilities

One of the fallouts of the development of the DAIS Executive from the Application Programmer's point of view was with respect to the ability to test the Application Software in a convenient and efficient manner. Often in the development of real-time software, and avionics software in particular, there is but one unique hardware facility available; the target processor. In addition, projects often have the hardware system under parallel development along with the OFP software. Not only do these hardware facilities have extremely limited tools available to the programmer, but access is limited to one user at a time. The hardware personnel also have priority over software development. In such circumstances, software personnel lose time and productivity. When the system does become available it has the characteristics of a unique system with respect to setup, initialization and tool usage: the programmer must learn yet another system.

One of the benefits of using a HOL is that the programs written with the HOL are no longer tied to a given computer but rather are portable. It becomes possible to write and

test routines on another computer, e.g., a large host computer, thus avoiding the problems of availability and training on a unique small system. However, real-time programming also involves the interaction of real-time processes, events, and time itself. There is seldom a convenient way to "model" the execution of the target computer on a large host, nor is there usually time or money to generate a viable simulator.

The availability of a set of real-time interface primitives alters this situation, by predicating what the executive interface must be, and thus making the Application Software invariant, not only with respect to the processor but also with respect to executive implementation. Further, if the process control for the executive is itself written in a HOL, then this executive has itself achieved portability. When the executive is written in a HOL, it becomes extremely simple to have a simulation capability on a host computer, assuming an appropriate code generator exists.

In the modern world of sophisticated flight processors, the ability to make effective use of an Interpretive Computer Simulator (ICS) has lessened. The execution time for the simulated computer versus the host computer is often several thousand to one. Thus, to simulate one second of target computer time could take over one thousand seconds: an untenable condition for real-time program development.

An alternative to this situation would be to use a Statement Level Simulator (SLS). This concept is based upon the following factors:

- The Program is being written in the specified HOL - The actual machine code is not being tested, but rather only the HOL implementation.
- The HOL has a Real-Time Executive Interface - This allows for a method of Application Software invariance with respect to executive implementation.

- A Host and Target Computer situation exists - There is available a convenient large host computer system but a relatively inaccessible target computer for software development.

In this context, it is possible to treat the HOL statement as the basic computation step and to calculate the amount of time it would require on the target computer. If code is now generated for the HOL program on the host computer, and after each statement the "Target Computer Statement Time" is added to a pseudo-clock, then the program can be executed on the host computer yielding the target computer timings and displaying the correct real-time interactions for simulation purposes. Thus a simulation facility can be provided which is of the order of two to one real time.

In order to implement the timing characteristics in the above scheme, compiler cooperation is necessary. HAL/S implemented such a mode of operation, named FSIM, and is currently in use in the development of the Space Shuttle OFP. JOVIAL J73/I has also an SLS mode, but this was not operationally available during the development of the current DAIS Mission Software.

In lieu of such a simulation capability, a Module Based Simulator (MBS) approach was used with great success. This mode of simulation is identical to be the SLS concept except the statement timings are unavailable. Thus the MBS runs as if it had an infinitely fast CPU available. Tasks run until they either complete or are waiting for I/O, events, or time.

Use of the MBS allowed DAIS Mission Software to be developed and functioning on the host computer system, the DECsystem-10, months before the availability of the actual flight computers, the AN/AYK-15. The usefulness of the method can be illustrated by the fact that the DAIS weapon delivery software was solely developed and tested in the DECsystem-10 and

then moved to the AN/AYK-15s. This was successfully accomplished in less time than two hours even though it involved a 10,000 word program.

The important fact to emphasize however, is that the MBS is inherently present under the design methodology used by the DAIS Executive. The executive primitives themselves form a simulation control language. Using the HOL and the Process Control Statements, it is possible to obtain complete control of the (pseudo) real-time interactions for tracing, dumping, analysis, and other simulation interactions. An additional advantage is that the Applications Programmer is already familiar with the simulation control language: it is the same language in which he has been programming. An MBS allows for the (pseudo) real-time development and testing of OFP's on a host computer independent of the actual flight systems. By the time the Mission Software is to be transferred to the target machine, logical, algorithmic, data, and real-time interactions should have been debugged and the program considered correct. What is left upon transfer to the target computer is to verify low level interfaces, system saturation characteristics, and actual system performance.

#### 5.2.2 Simulation Method

Development of OFP's in Higher Order Languages (HOLs) is an improvement over assembly language methods; however, there are still many problems associated with OFP development. One of the most significant problems is lack of availability of the Flight Computer. A solution would be to have OFP validation completed prior to Hot Bench Computer (HBC) testing. Checkout could be made by three different methods. The first method is to perform module-type checkout where a given set of inputs to a module result in predetermined outputs. This type of checkout is very minimal and while providing for the correctness of a given module, it does not verify the system interaction. The second type of checkout is performed using

a complete system which includes both an OFP and simulation modules. This type of checkout is performed non-real time and will provide for the checkout of the complete OFP as a system (with all inputs from the simulated real world and outputs to simulated sensors). This will verify correctness of those items subject to mathematical analysis. The third method of OFP checkout is an extension of the second method where a Head Up Display (HUD) is implemented and dynamic inputs from a cockpit are used. This is the final type of system checkout before the HBC is used. This is required in order to assure correctness with respect to visual human factors (e.g., Bomb fall line).

Each of these three methods of OFP checkout were used for the DAIS Mission Software. The module type is used as a preliminary test to discover any blatant errors. After the modules are individually exercised, they are integrated and the second method is used. Figure 20 indicates the structure of the second and third method of OFP checkout, and integration with the models. The second method does not require the use of the Evans-Sutherland display system or the cockpit. Instead the cockpit inputs are performed via keyboard input to the DECsystem-10. However, it should be noted that the OFP inputs from the models are identical to those that would arrive from the URT when the OFP is resident in the HBC. This second method assures complete model and OFP checkout at the systems level. The current simulation method uses the TOMBS simulation facility, the AVSIM Model Executive, and the DECsystem-10 DDT facility. Data may be logged from both the models and the OFP for post run analysis. The third method of checkout allows the programmer to evaluate the dynamic system interaction of the OFP and the models. The programmer may view the Evans-Sutherland and verify that the dynamic displays and logic are performing as desired. Data may again be logged from the OFP and models for post run analysis. Thus, for example, Miss and Release calculations can be evaluated later from their logged values. The current simulation implementation has a freeze button that will halt

the models and OFP in order to allow the DDT facility to view all the Compools in the OFP and all Common blocks in the models.

As shown in Figure 20, the third method makes use of the cockpit and the Evans-Sutherland system. Once method three has been completed, the OFP is cross-compiled for the HBCs and the final phase of verification is performed.

A further capability to this system is that the input data to the OFP can be logged on tape, and then later it can be used as input to the URT when the OFP is resident in the STS or ITB. A direct comparison of the HBC's output against the previous DECsystem-10 output will verify accuracy and validity of the HBC resident OFP.

These techniques not only allow the programmer to progress through higher levels of testing in a sequence of logical steps, but they also provide a means of assuring that the models and the OFP perform together in a prescribed manner. When the OFP is transferred to the HBC very minimal control is allowed and very minimal I/O is available for the evaluation of the model and OFP interactions.



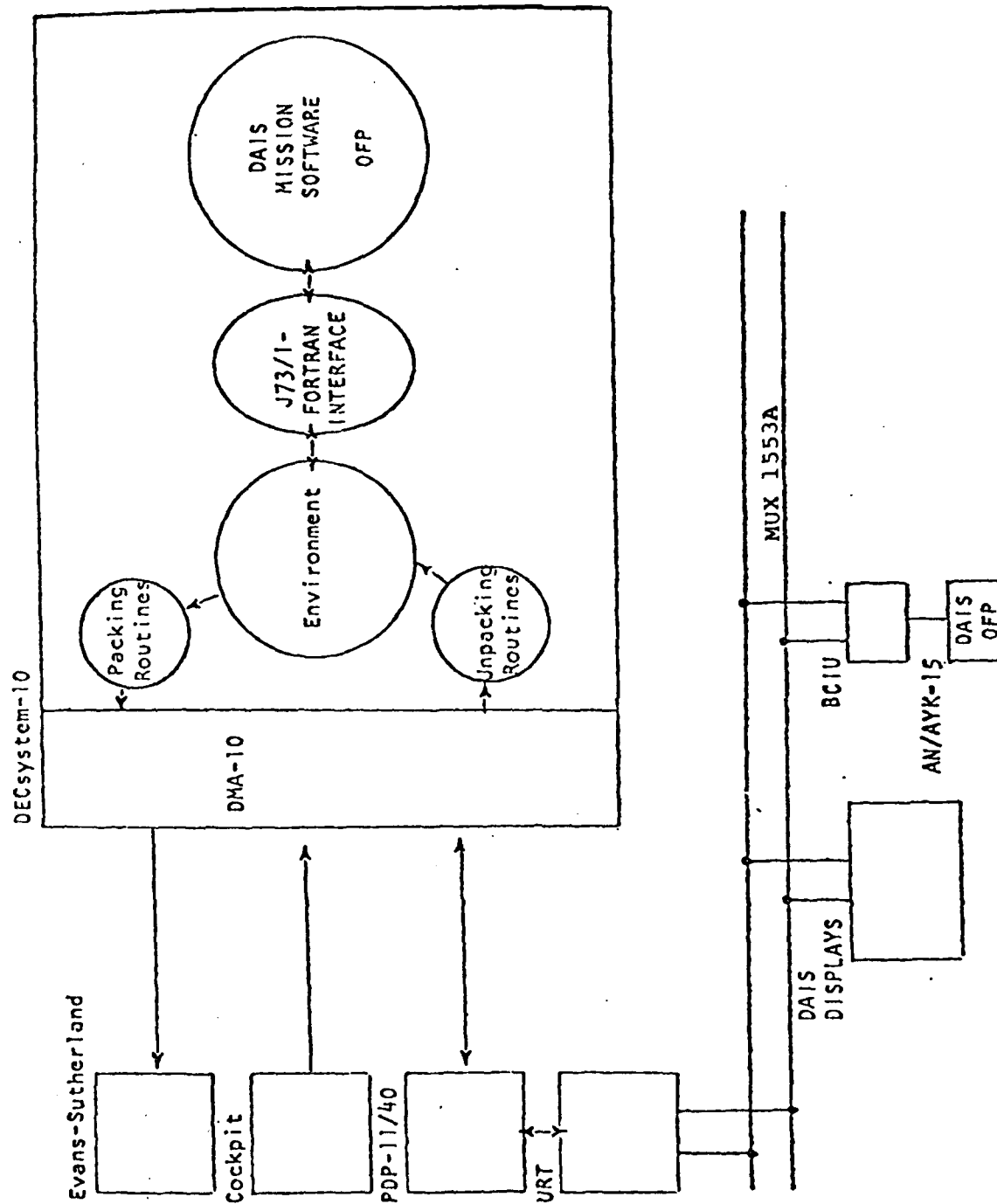


Figure 20. SYSTEM CHECKOUT CONFIGURATION

### 5.2.3 SIMULATION TOOLS

Simply stated, the object of software testing is to assist in the development, assure the correctness, and measure the performance of the software system to be delivered. As the complexity of a system grows, so does the need for a well-designed method of proving that the system works.

Just as module-based block-structured programming aids the coding of large systems by reducing the conceptual complexity to smaller, more manageable problems, a module-based approach to software testing provides a systematic solution to the difficult problem of proving system correctness. The tests should be designed so as to minimize the number of potential sources of error at any one testing step, thus eliminating confusion when discrepancies do appear.

Applying this reasoning to the DAIS system, a four phase testing process suggests itself. The first two phases apply to individual program modules.

Initially, a module is tested in a static environment, verifying the response of the module's algorithms to inputs. Once this test is passed, the module may be subjected to dynamic tests, verifying its performance within the structure provided by the local executive (application software) or other executive routines (executive software).

Intermetrics provided several tools to aid in this preliminary testing. The Level 0 tester (L0GEN) sets up static tests and generates test reports. The display system requires its own specialized Level 0 Tester (IMFKT). The TOMBS facility provides a local executive environment for dynamic module-based testing. These tools are described below in more detail.

Once the modules are individually verified, integration with the AVSIM simulation may occur, executing both the DAIS system and the real time environmental simulation on the DECsystem-10. This provides all the services of the DECsystem-10 debugging tools rendering visibility to any problems arising from this integration step.

Finally, the tested DAIS system must be moved to the ITB. Any problems occurring there may be attributed to the peculiarities of the ITB with confidence that the algorithms and control structures of DAIS and the DAIS communication with the real time simulation are valid.

The testing methodology espoused above therefore consists of the following sequence:

- Level 0: Individual Module static test
- Level 1: Individual Module dynamic test with local executive
- Level 2: Integrated DAIS System with Simulator on DECSYSTEM-10
- Level 2A: Full-up System execution on HBC

#### 5.2.3.1 General Level Ø Testing

The DAIS Software Standards enforced the differentiation between input, output, and update variables. Thus, it was possible to build a tool which automatically generates a Level Ø test driver for the DAIS Mission Software Tasks and COMSUBS.

LØGEN is an automated facility for performing Level Ø tests of individual applications software tasks. It uses the source file of an applications task to automatically write a set of test programs and input files for a Level Ø test. These files are automatically compiled and linked to run under TOMBS.

The user then executes this load file. The test program is run interactively on a computer terminal. The user is requested by the test program to enter a value for each input variable and event of the task. These values are output to the task, the task itself is executed, and all output values of the task are printed on the user's terminal. The output includes values for all output variables, events, and a list of any tasks that have been scheduled by the task being tested. The user may continue running the test as many times as desired. In subsequent runs, he may reset each input variable or leave it set to the value at the end of the preceding run.

While the user is inputting variable values and the test program is outputting values through the terminal, a separate report file is kept documenting the user's inputs and the test programs' outputs in printable form. Thus the Level Ø tester allows the user to start with the source file of a task and generate a test report in a minimal amount of time. The only entries he must make are the task name and the input variable values when asked for.

#### 5.2.3.2 Level 0 Testing of Display Software

IMFKT is similar in concept to LØGEN but has been specialized for testing the OFP Display Software. This tool will print out the sequence of display messages in visual form retaining the actual final display image in addition to the above mentioned Level 0 functions. A report file is also generated logging all terminal interactions in a printable form.

#### 5.2.3.3 Level 1 and Level 2 Testing

TOMBS (Tony's Operational Module-Based Simulator) is a simulation facility executing on the DECsystem-10. TOMBS is similar to an SLS (Statement Level Simulator) in that it executes host computer instructions while emulating target computer (e.g., AN/AYK-15 Processor) programs.

This facility was the primary tool used in conjunction with AFAL's AVSIM for the initial development and testing of the navigation and weapon delivery software for the DAIS Mission Software effort. Upon successful testing in the DECsystem-10, the transfer to the AN/AYK-15s by recompilation has proven to be a simple and straightforward process, with the major source of error being with respect to interface discrepancies.

##### 5.2.3.3.1 TOMBS

TOMBS is a module-based simulator that will allow real time interaction to occur at the end of each module and/or at the occurrence of the HOL's real time statements (i.e., executive interfaces). Therefore, TOMBS presents a simulation facility that is capable of accurate Real Time Interaction with at least the fidelity and granularity of the module (task) level. Its throughput capability will exceed that of an SLS.

The implementation of TOMBS drives the system with respect to minor cycle interrupts. TOMBS does not make any assumptions with respect to the timing, or the accumulation of time, by the executing program. New minor cycles are generated by TOMBS when all Application Tasks are in WAITING or INACTIVE states. Thus, TOMBS simulates a situation in which the flight computer is considered to be infinitely fast. All tasks run to completion in a finite amount of time, and in particular, in a Minor Cycle.

From the point of view of TOMBS, an Applications Task can be suspended only at Real Time statements. From the point of view of the Applications Software, TOMBS is supporting a time granularity of a Minor Cycle. Note, however, that this is the finest time granularity allowed and supported for Task interactions in the DAIS Mission Software System. Only the time critical output statement TRIGGER allows for a finer (relative) time granularity in the DAIS Mission Software System. But the execution of the TRIGGER statement itself cannot be finer than a minor cycle.

The DAIS Applications Software is to be designed and built as if it were to reside in a single processor. The fact that DAIS is a Federated computer system and requires software partitioning should be invisible to the Applications Software. Indeed, a major design emphasis with respect to the DAIS Executive and Software Standards is precisely that development methodology.

TOMBS does not in itself implement a specific set of simulation control facilities. What it does, however, is to allow the use of the JOVIAL debug and I/O capabilities. In addition, TOMBS presents a framework in which the user may write his own control and environmental program which are then easily associated with the Applications Software to be tested.

#### 5.2.3.3.2 TOMBS and AVSIM

AVSIM consists of a set of avionics models and a structure for controlling them. This facility is the baseline in use at AFAL for A-7, A-10, and F-16 simulations. AVSIM is used in conjunction with the DAIS STS and ITB facilities.

TOMBS has been directly interfaced with AVSIM in order to allow a total simulation environment for the execution of mission software. This interfacing presents the same models and timing interactions as found when executing on the actual processors in the STS or ITB environment.

## SECTION VI. MISSION SOFTWARE ILLUSTRATIONS

During the three and a half year duration of the DAIS Mission Software program, the mission software and the DAIS program itself continued to evolve and assume new capabilities and directions. Part of this evolution arose in response to other Air Force programs. The changes to the DAIS Mission Software as a result of this evolution illustrates the capability of the basic design and methodology to quickly respond to dynamic revisions to requirements.

This section will discuss some of these developments and the ability of the DAIS Mission Software to respond.

### 6.1 Mission A to Mission $\alpha$

With the evolution of the DAIS program, the baseline A-7 aircraft mission was redesignated to be an A-10 mission. In addition, the actual equipment suite was modified with respect to both sensors and the inertial system. This, of course, had a corresponding effect upon the control and display requirements.

In Section 4.5 of this report, the structuring of the Application Software was discussed. When the change was made from the A-7 based Mission A to the A-10 based Mission  $\alpha$ , the baseline OFP had to be modified. The structural nature of the Application Software in each of its functional areas: control programs; navigation, guidance and control; weapon delivery; stores management; pilot interface; and equipment interfaces; allowed for rapid and reliable modification to the new mission and its requirements.

An example of the modifications required can be seen in the navigation, guidance, and control functions. Navigation was structured initially on DAIS for Mission A to include



all the functions necessary for maintenance of an inertial platform. This included processing accelerometer outputs supplying gyro torquing, and performing ground alignments.

With the purpose of providing a system which would contain all the capabilities of the existing A-7 OFP, Navigation was able to use observables from the Doppler radar to achieve in-flight alignment, baro-altimeter data for vertical channel damping, and contained fault-down logic if one or more of the sensors failed. The original navigation function also provided air-data and winds computation as well as position updates and aircraft attitude processing.

When DAIS was re-directed to use the SKN-2416 Inertial Navigation System for Mission  $\alpha$ , much of the navigation was rendered unnecessary, as the INS did internal accelerometer, gyro, and alignment maintenance. The necessary software modifications were very minimal, however. Due to the block-structure of the navigation function, the unnecessary portions of the OFP were simply removed, leaving the functions of position, air-data, winds, and attitude processing virtually unchanged. The simplicity with which the modification was made is indicative of the value of the layered modular software approach.

The Guidance function had been structured in a similar fashion. Modules for different modes of horizontal and vertical guidance were made with no significant modifications to the algorithmic content of existing software. Changes occurred only in the controlling and moding logic of the Steering controller.

## 6.2 Non-DAIS Device Protocol

Early in the DAIS program, AFAL decided to interface its system to the ADTC Stores Logic Units (SLU) for the Stores Management System. The SLU interface design, however, did not follow the DAIS MIL-STD 1553A protocol in detail. Initially, this was treated as a special condition in the I/O handling.

Associated with the modification of the mission for DAIS, was the introduction of the SKN-2416 inertial system. While this system is used on the F-16 and interfaces with a MIL-STD 1553 type protocol, it does not use the DAIS protocol in detail. The Single Seat Attack (SSA) program at the time was also based upon an A-10 and had selected a SKN-2416 inertial unit for its system. The SSA program also chose to use the DAIS Executive system and methodology for its Mission Software development.

In addition to not following the DAIS MUX protocol, the SKN-2416 required the ability for the application software to read the device whenever it wanted. Previously, I/O was either read to the Application Software in an Asynchronous periodic manner, or asynchronously when the device indicated it needed to be read. As a result of the above requirement, the DAIS Software System was extended to be able to handle (1) Non-standard Avionics (i.e., different protocols), and (2) Application Software Asynchronous device reads.

Non-Standard devices, in the context of DAIS, are those devices capable of communication over a MIL-STD 1553A bus but which do not conform exactly to the DAIS Remote Terminal (RT) communications protocol. Since the DAIS Bus Control Executive was designed to communicate with DAIS compatible Remote Terminals only, modifications were required to support such non-standard devices. Three components of DAIS Executive Software were affected by the required modification: PALEFAC, the Local Executive, and the Bus Control Master Executive.

The capability of an applications task to asynchronously read data from a terminal connected to the bus departs from the baseline DAIS system philosophy in that terminals connected to the bus may only asynchronously transmit when they wish to, never on demand. When an applications task reads a data block, it only accesses the last updated version of the data from the processor's memory, and in no way forces a transmission from the related device.

Since the new capability differed radically from the previous concept of READ, a new real-time linguistic primitive to perform the operation was invented, entitled FORCE'READ. This primitive would make a special request to the bus control to update the Compool block argument by forcing a transmission from its associated terminal. The Local Executive would then place the issuing task into a wait state until the Compool block was updated. This time suspension was required due to the time lag involved in bus transmissions. This augmentation to the real-time Applications/Executive interface required modifications to PALEFAC for the recognition of the new primitive. Appropriately revised table constructions and corresponding local Executive interface routines were created as a result of the modification.

### 6.3 Throughput Optimization

The rigidity of the Higher Order Software (HOS) principles insures that real-time data conflicts can not exist. The implementation of these principles requires double buffering of data to achieve this condition. Global information is read into a local copy, which is manipulated, and finally the local copy may be written back to the global copy. The reading or writing of these copies occur as a complete function that cannot be interrupted by other data moves. (Although hardware interrupts can occur, data interface is not allowed.) While this method of preventing data conflicts works, it obviously entails a high execution time penalty.

One of the first modifications to HOS principles was the introduction of a GLOBAL'COPY data declaration along with the associated ACCESS and BROADCAST real-time statements. These three primitives correspond to the LOCAL'COPY declaration and the READ and WRITE statements. However, with the GLOBAL'COPY mechanism, access to the actual global data is allowed without the double buffering. In order to ensure that reliability is maintained, these global primitives may only occur in the

highest priority processes, and thus non-interruptable tasks, that is, these new declarations and statements, may only occur in PRIVILEGED'MODE'TASKS which are short and non-interruptible tasks.

While these restrictions prevent data conflicts, the majority of tasks still use the LOCAL'COPY, READ and WRITE primitives. It is possible, however, to analyze and understand possible data conflicts and to ensure that data blocks cannot have such conflicts. Therefore, a new construct, the LOCAL'COPY'OVERRIDE directive, was introduced as an efficiency capability to be used only with care. The use of this directive, effectively modifies the code to behave in the same manner as the new global primitives while still being written as LOCAL'COPY, READ and WRITE. It was used both by the SSA program and by the DAIS Weapon Delivery Software in its two processor demonstration. When used in a careful and knowing fashion, the directive can be of assistance. But it also assumes an *a priori* guaranteed data conflict prevention mechanism and cannot be used indiscriminately.

#### 6.4 Proof of Concept

The DAIS Mission Software was developed and tested primarily on the host computer, the DECsystem-10. Only after the Mission Software was developed, simulations completed, and integrity assumed was the Mission Software placed on the flight processors, the AN/AYK-15's. The initial Mission A Application Software and Local Executive were developed in their entirety on the DECsystem-10 before the physical AN/AYK-15s arrived at AFAL. While the framework of the Master Executive's bus control existed, it required testing of the AN/AYK-15 hardware interfaces to assure proper functioning. The not unusual situation of limited target computer access, few support tools, and continued system hardware integration prevented the full use of the DAIS STS and ITB during this effort.

Throughout the DAIS Mission Software efforts approximately 75% of the debugging and testing of the Local and Master Executives was accomplished on the DECsystem-10, with only the final processor and I/O interfaces, and virtual timing being of necessity performed on the actual AN/AYK-15 system. When the full weapon delivery capability of Mission α was developed, the effort was done on the DECsystem-10 using the old AFAL F-111 cockpit and displays. Not only were the logical and algorithmic capabilities debugged and tested, but the necessary visual weapon delivery human factors were verified before moving to the AN/AYK-15 processors and DAIS cockpit. The moving of this software and its integration into the target processor environment was accomplished in less than two days. This included recompilation of the programs, relinking, partitioning, and integration into the STS.

One of the most dramatic demonstrations of the DAIS Mission Software capability was the repartitioning of Mission α from a two processor demonstration into a three processor partitioned demonstration. This was accomplished in two hours. The philosophy and methodology of the development of the Applications Software with the basic design of the Executive System, allowed this base for simplified software integration.

#### 6.5 Use of J73/I

One of the objectives of the DAIS Mission Software effort was to use the Higher Order Language J73/I for the avionics software. The Application Software was successfully written using J73/I as controlled by the DAIS Software Standards. It was expected that 100% of the Application Software could be written in J73/I and this was accomplished. Certain data packing and unpacking routines could have been written in a more efficient manner in assembly code. Part of the reason for this was the fact that J73/I (as implemented on the AN/AYK-15 flight processor) only supported 16 bit integers rather than the 32 bit integers available on the processor hardware. Thus certain of the data manipulation became cumbersome in J73/I.

The DAIS Local Executive is responsible for interfacing with the Application Programmer. As such it provides not only process control, but also presents the interface for data handling and the I/O control. The Local Executive could be thought of as a set of services that modify the process state database by Application Software request.

The DAIS Local Executive was written 95% in J73/I with only three functions being coded in assembly code. These three functions are:

- The actual I/O instruction.
- The Interrupt Interface.
- The process swapping.

In each of these three cases, the feature to be executed is not conceptually supported by J73/I or other common HOLs, but are dependent upon hardware idiosyncrasies.

The DAIS Master Executive is concerned with Minor Cycle synchronization, message processing, and bus failure control. The messages may be either synchronous, asynchronous, or critically-timed, i.e., messages for the special TRIGGER statement. Over 90% of the Master Executive was written in J73/I. As was the case with the local executive, assembly code was logically required for the hardware and I/O interfaces.

The J73/I compiler implementation used in DAIS was not highly optimized, but the Mission Software functions implemented mapped well into the J73/I language. Thus the efficiency of code generation becomes a question of the J73/I language mapping into a particular instruction set.

#### 6.6 Embedded Performance Monitor

An embedded performance monitor (EPM) was implemented as an optional feature of the DAIS Executive. The motivations for such a tool were several. A requirement existed for a performance monitor embedded within the Executive itself. These requirements included:

- Software to provide constant monitoring without specific manual intervention.
- Software tailored to produce the desired results directly instead of trying to cull the information from massive post-run dump files.
- The Software that can perform this function in the absence of sophisticated debugging facilities, such as at field sites or even during flight.
- Performance parameters that would supply a uniform terminology for evaluating performance and as such fit in with the standardization effort promoted by DAIS.

While being directly embedded in the DAIS Executive, the Embedded Performance Monitor (EPM) is an optional element selected by conditional compilation features. Within the EPM in turn, various options are available to control the extent of monitoring to be performed. The option consists of three levels of information. These are:

1. Global Information - Measurements included in this area include total Executive, Applications, and Idle execution time.
2. Functional Area Information -
  - a. Interrupt processing time, number of interrupts.
  - b. Bus control time.
  - c. Bus throughput data:
    - i. number of messages
    - ii. estimated DMA conflict percentage
  - d. Task control time (count of real-time operations).
  - e. Timer control time (minor cycle setup, critically timed messages).
  - f. Application task execution time.
  - g. Other specific functional areas as desired.

3. Module Information - Measures timing for each module desired.

The above statistics can be measured and averaged over any duration. They supply data for the purposes of evaluating benchmarks or actual OFP performance. The overhead introduced by performance monitoring itself can be cancelled by halting the clock during performance processing. When real-time use is required, clock stopping cannot occur, but time spent in performance monitoring can be tallied separately from all other categories.



SECTION VII.  
" CONCLUSION AND RECOMMENDATIONS

The DAIS Mission Software was successfully developed and it demonstrated and proved that the use of modern software techniques and reliability concepts are effective in the rigorous environment of avionics. The capabilities of J73/I as a real-time programming language has been demonstrated and upward compatible real-time linguistic constructs for use with J73/I have been identified. The use of modern techniques has allowed the development of easily partitionable mission software. Indeed the Mission Software was developed as a single virtual system and then partitioned as desired. Finally, the real-time interface allowed the efficient and cost-effective use of simulation techniques.

RECOMMENDATIONS

The achievement of the DAIS Mission Software has opened avenues for further development and refinement. The following constitutes a list of items requiring further development or refinement and several recommendations:

- One of the major drawbacks in the software community acceptance of the DAIS Executive is the question of time and space efficiency. In order to be accepted by a broad based community, it is necessary to maintain the efficiency of the DAIS system without interfering with the inherent reliability of its development approach and without increasing the complexity of its interface from the Applications Software point of view. In particular, an improved implementation of data blocks (READ/WRITE) must be addressed. The proper method of optimization would be the use of PALEFAC for the analysis of conflict situations. This area represents a new stage of technology.

- The ground work and preliminary design for a concept of Monitor Recovery has been laid. These efforts should be implemented and demonstrated. It is important to maintain the philosophy of creating a framework in which missions are easily developed and OFPs are modifiable in a cost effective manner.
- Improvements to the current Mission Software Build process are required. In particular, the relationship between PALEFAC, the PALEFAC Pre-processor, and the Application modules and Compoools need to be improved by automating the necessary creation and transitioning of information.
- PALEFAC provides a central repository of information. While it has always been intended for PALEFAC to analyze this information for diagnostics, statistics and enforcement of coding standards, this has as of yet been done only to a small degree. This should be pursued in order to fully develop the tool and in order to provide useful statistics for future Air Force efforts.

Perhaps the one area that has not yet been fully appreciated in the DAIS program is the generality of the DAIS software system.

Current avionics systems such as the F-16 or F-18 are actually federated computer systems with many large and small computers. While the F-16 is advertised as having one Fire Control Computer, it also contains processors in its radar, HUD, and other subsystems.

Once DAIS has been retargeted to systems other than the AN/AYK-15, it will be possible to accommodate mixed systems of processors within the DAIS federated scheme. To ensure a reasonable and desirable goal the required interfaces must

match. Since J73/I is the linguistic expression of DAIS on a high level, it only becomes necessary to ensure compatibility on the low level. This implies:

- Use of the same (compatible) data bus and protocol
- Identical organization and structure of Interprocessor messages
- Invariant floating point format between processors (e.g., 48-bit representation: 16 bit exponent and 32 bit mantissa in 2's complement notation.)

By requiring this compatibility for all versions of the Executive/PALEFAC it will become possible to execute the same PALEFAC input with the same set of Applications Software on each version of PALEFAC to obtain compatible load modules for differing processors.

In the same vein even small processors (8080s, 6800s) for which J73/I code generators exist, can be centrally developed and then partitioned out by PALEFAC. In this case, the target processors would not have a full executive capability. But the advantages of a central development and control would be realized. This concept is important in the context of the emerging and growing use of microprocessors in avionic systems and the associated development of distributive processing. Thus the distinction between central processors and remote subsystems would cease to be one of software versus hardware functions.

The applicability of the DAIS Software system to mixed processors of varying types and sizes should be demonstrated in the context of central software development and/or control. The current DAIS Software system forms a baseline that is easily extended in this direction.

With the DAIS Mission Software, it is possible to transition more than techniques and methodology. It is possible also to transition the support tools and the avionics executive.

Because of this capability, it becomes extremely critical for the Executive to be designed, implemented, and verified in such a fashion as to both claim and receive user acceptance. Major life cycle cost savings are realizable by standardizing the Executive/Application Software interface. If this interface (the real-time constructs) are standardized, then it is relatively straightforward to build or retarget a DAIS executive for various avionics computers or I/O structures.

If the Real-Time interface were to be accepted by the Air Force as a standard it would no longer be possible to continue major developments or modifications. It is imperative that all major outstanding issues be resolved in a timely fashion while maintaining system reliability and integrity.

### Bibliography

The following is a list of significant documents generated under this contract.

PA200101, DAIS Software Development Standards; Intermetrics Report #140.

SA201302, DAIS Mission Software Executive Specification; Intermetrics Report #139.

SA201302, DAIS Mission Software Product Specification, Volume 1: Local Executive; Intermetrics Report #147/1.

SA201302, DAIS Mission Software Product Specification, Volume 2: Bus Control; Intermetrics Report #147/2.

DAIS Mission Software Executive User's Guide; Intermetrics Report #157.

SA201303, DAIS Mission Software Design Specification: Operational Flight Program Applications; Intermetrics Report #141.

SA201305, Computer Program Product Specification for DAIS Mission Software Operational Flight Program Applications; Intermetrics Report #158.

User's Guide, TOMBS Module Based Simulation; Intermetrics Report #145.

